MPC8349EA PowerQUICC™ II Pro Integrated Host Processor Family Reference Manual

Supports

MPC8349EA
MPC8347EA
MPC8343EA

MPC8349EARM
Rev. 1
08/2006

freescale™
semiconductor

How to Reach Us:

Home Page:
www.freescale.com

email:
support@freescale.com

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

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

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

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

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

Document Number: MPC8349EARM
Rev. 1, 08/2006

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™ and the Freescale logo 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.

Portions of Chapter 19, “USB Interface,” relating to the EHCI specification are Copyright © Intel Corporation 1999–2001. The EHCI specification is provided “As Is” with no warranties whatsoever, including any warranty of merchantability, non-infringement, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification or sample. Intel disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in the EHCI specification. Intel may make changes to the EHCI specification at any time, without notice.

© Freescale Semiconductor, Inc. 2005, 2006.

BUILT ON Power™

freescale™ semiconductor

Preface PRE
Overview 1
Memory Map 2
Signal Descriptions 3
Reset, Clocking, and Initialization 4
System Configuration 5
Arbiter and Bus Monitor 6
e300 Processor Core 7
Integrated Programmable Interrupt Controller (IPIC) 8
DDR Memory Controller 9
Local Bus Controller 10
Sequencer 11
DMA 12
PCI Bus Interface 13
Security Engine (SEC) 2.0 14
Three-Speed Ethernet Controllers 15
Universal Serial Bus Interface 16
I^2C Interfaces 17
DUART 18
Serial Peripheral Interface 19
JTAG/Testing Support 20
General Purpose I/O (GPIO) 21
Delay Lock Loop (DLL) 22

Revision History APP
Glossary GLO
Register Index REG
General Index IND

PRE Preface

1 Overview

2 Memory Map

3 Signal Descriptions

4 Reset, Clocking, and Initialization

5 System Configuration

6 Arbiter and Bus Monitor

7 e300 Processor Core

8 Integrated Programmable Interrupt Controller (IPIC)

9 DDR Memory Controller

10 Local Bus Controller

11 Sequencer

12 DMA

13 PCI Bus Interface

14 Security Engine (SEC) 2.0

15 Three-Speed Ethernet Controller

16 Universal Serial Bus Interface

17 I^2C Interface

18 DUART

19 Serial Peripheral Interface

20 JTAG/Testing Support

21 General Purpose I/O (GPIO)

22 Delay Lock Loop (DLL)

APP Revision History

GLO Glossary

REG Register Index

IND General Index

Contents

Paragraph Number	Title	Page Number

About This Book

Audience lxxv
Organization lxxv
Suggested Reading lxxvii
Conventions lxxviii
Signal Conventions lxxix
Acronyms and Abbreviations lxxix

Chapter 1
Overview

1.1 MPC834x-Family Product Distinctions 1-1
1.2 MPC8349EA PowerQUICC II Pro Processor Overview 1-2
1.3 MPC8349EA Architecture Overview 1-7
1.3.1 PowerPC Core 1-7
1.3.2 Security Engine 1-10
1.3.3 DDR Memory Controller 1-10
1.3.4 Dual Three-Speed Ethernet Controllers 1-11
1.3.5 PCI Controllers 1-12
1.3.5.1 PCI Bus Arbitration Unit 1-12
1.3.6 Universal Serial Bus (USB) 2.0 1-12
1.3.6.1 USB Dual-Role Controller 1-13
1.3.6.2 USB Multi-Port Host Controller 1-14
1.3.7 Local Bus Controller (LBC) 1-14
1.3.8 Integrated Programmable Interrupt Controller (IPIC) 1-15
1.3.9 Dual I^2C Interfaces 1-15
1.3.10 DMA Controller 1-16
1.3.11 Dual Universal Asynchronous Receiver/Transmitter (DUART) 1-16
1.3.12 Serial Peripheral Interface (SPI) 1-17
1.3.13 System Timers 1-17
1.4 Applications 1-18
1.5 Differences Between MPC8349E and MPC8349EA 1-18

Contents

Paragraph Number	Title	Page Number
	Chapter 2 Memory Map	
2.1	Internal Memory Mapped Registers	2-1
2.2	Accessing IMMR Memory From the Local Processor	2-1
2.3	Complete IMMR Map	2-1
	Chapter 3 Signal Descriptions	
3.1	Signals Overview	3-1
3.2	Configuration Signals Sampled at Reset	3-19
3.3	Output Signal States During Reset	3-19
	Chapter 4 Reset, Clocking, and Initialization	
4.1	External Signals	4-1
4.1.1	Reset Signals	4-1
4.1.2	Clock Signals	4-3
4.2	Functional Description	4-3
4.2.1	Reset Operations	4-3
4.2.1.1	Reset Causes	4-4
4.2.1.2	Reset Actions	4-4
4.2.2	Power-On Reset Flow	4-5
4.2.3	Hard Reset Flow	4-7
4.2.4	Soft Reset Flow	4-8
4.3	Reset Configuration	4-8
4.3.1	Reset Configuration Signals	4-9
4.3.1.1	Reset Configuration Word Source	4-9
4.3.1.2	CLKIN Division	4-10
4.3.1.3	Selecting Reset Configuration Input Signals	4-10
4.3.2	Reset Configuration Words	4-11
4.3.2.1	Reset Configuration Word Low Register (RCWLR)	4-11
4.3.2.1.1	System PLL Configuration	4-12
4.3.2.2	Reset Configuration Word High Register (RCWHR)	4-14
4.3.2.2.1	PCI Host/Agent Configuration	4-16
4.3.2.2.2	Boot Memory Space (BMS)	4-16
4.3.2.2.3	Boot Sequencer Configuration	4-17
4.3.2.2.4	Boot ROM Location	4-17
4.3.2.2.5	TSEC1 Mode	4-19

Contents

Paragraph Number	Title	Page Number
4.3.2.2.6	TSEC2 Mode	4-20
4.3.2.2.7	e300 Core True Little-Endian	4-20
4.3.2.2.8	LALE Configuration	4-21
4.3.2.2.9	LDP Configuration	4-21
4.3.3	Loading the Reset Configuration Words	4-21
4.3.3.1	Loading from Local Bus EEPROM	4-21
4.3.3.1.1	Local Bus EEPROM Timing	4-23
4.3.3.2	Loading from I2C EEPROM	4-24
4.3.3.2.1	Using the Boot Sequencer Reset Configuration	4-24
4.3.3.2.2	EEPROM Calling Address	4-24
4.3.3.2.3	EEPROM Data Format in Reset Configuration Mode	4-24
4.3.3.2.4	Reset Configuration Load Fail	4-27
4.3.3.3	Default Reset Configuration Words	4-27
4.3.3.3.1	Hard-Coded Reset Configuration Word Low	4-27
4.3.3.3.2	Hard-Coded Reset Configuration Word High Fields Values	4-28
4.3.3.3.3	Examples for Hard-Coded Reset Configuration Words Usage	4-29
4.4	Clocking	4-29
4.4.1	Clocking in PCI Host Mode	4-30
4.4.1.1	PCI Clock Outputs (PCI_CLK_OUT[0:7])	4-30
4.4.2	Clocking In PCI Agent Mode	4-30
4.4.3	System Clock Domains	4-30
4.5	Memory Map/Register Definition	4-32
4.5.1	Reset Configuration Registers Descriptions	4-32
4.5.1.1	Reset Configuration Word Low Register (RCWLR)	4-32
4.5.1.2	Reset Configuration Word High Register (RCWHR)	4-32
4.5.1.3	Reset Status Register (RSR)	4-33
4.5.1.4	Reset Mode Register (RMR)	4-34
4.5.1.5	Reset Protection Register (RPR)	4-35
4.5.1.6	Reset Control Register (RCR)	4-36
4.5.1.7	Reset Control Enable Register (RCER)	4-36
4.5.2	Clock Configuration Registers	4-37
4.5.2.1	System PLL Mode Register (SPMR)	4-37
4.5.2.2	Output Clock Control Register (OCCR)	4-38
4.5.2.3	System Clock Control Register (SCCR)	4-40
4.5.3	Clock Control DDR Registers	4-41
4.5.3.1	MCK Enable Register (MCKENR)	4-41

Contents

Paragraph Number	Title	Page Number
	Chapter 5 System Configuration	
5.1	Introduction	5-1
5.2	Local Memory Map Overview and Example	5-1
5.2.1	Address Translation and Mapping	5-3
5.2.2	Window into Configuration Space	5-3
5.2.3	Local Access Windows	5-4
5.2.3.1	Local Access Register Memory Map	5-4
5.2.4	Local Access Register Descriptions	5-5
5.2.4.1	Internal Memory Map Registers Base Address Register (IMMRBAR)	5-5
5.2.4.1.1	Updating IMMRBAR	5-6
5.2.4.2	Alternate Configuration Base Address Register (ALTCBAR)	5-7
5.2.4.3	LBC Local Access Window *n* Base Address Registers (LBLAWBAR0–LBLAWBAR3)	5-7
5.2.4.3.1	LBLAWBAR0[BASE_ADDR] Reset Value	5-8
5.2.4.4	LBC Local Access Window *n* Attributes Registers (LBLAWAR0–LBLAWAR3)	5-8
5.2.4.4.1	LBLAWAR0[EN] and LBLAWAR0[SIZE] Reset Value	5-9
5.2.4.5	PCI Local Access Window *n* Base Address Register (PCILAWBAR0–PCILAWBAR1)	5-9
5.2.4.5.1	PCILAWBAR0[BASE_ADDR] Reset Value	5-10
5.2.4.6	PCI Local Access Window *n* Attributes Registers (PCILAWAR0–PCILAWAR1)	5-10
5.2.4.6.1	PCILAWAR0[EN] and PCILAWAR0[SIZE] Reset Value	5-11
5.2.4.7	DDR Local Access Window *n* Base Address Registers (DDRLAWBAR0–DDRLAWBAR1)	5-12
5.2.4.7.1	DDRLAWBAR0[BASE_ADDR] Reset Value	5-12
5.2.4.8	DDR Local Access Window *n* Attributes Registers (DDRLAWAR0–DDRLAWAR1)	5-13
5.2.4.8.1	DDRLAWAR0[EN] and DDRLAWAR0[SIZE] Reset Value	5-13
5.2.5	Precedence of Local Access Windows	5-14
5.2.6	Configuring Local Access Windows	5-14
5.2.7	Distinguishing Local Access Windows from Other Mapping Functions	5-14
5.2.8	Outbound Address Translation and Mapping Windows	5-15
5.2.9	Inbound Address Translation and Mapping Windows	5-15
5.2.9.1	PCI1/PCI2 Inbound Windows	5-15
5.2.10	Internal Memory Map	5-15
5.2.11	Accessing Internal Memory from External Masters	5-16
5.3	System Configuration	5-16
5.3.1	System Configuration Register Memory Map	5-16

Contents

Paragraph Number	Title	Page Number
5.3.2	System Configuration Registers	5-17
5.3.2.1	System General Purpose Register Low (SGPRL)	5-17
5.3.2.2	System General Purpose Register High (SGPRH)	5-17
5.3.2.3	System Part and Revision ID Register (SPRIDR)	5-18
5.3.2.3.1	SPRIDR[PARTID] Coding	5-18
5.3.2.4	System Priority and Configuration Register (SPCR)	5-19
5.3.2.5	System I/O Configuration Register Low (SICRL)	5-21
5.3.2.6	System I/O Configuration Register High (SICRH)	5-24
5.3.2.7	Debug Configuration	5-28
5.3.2.7.1	DDR Debug Configuration	5-28
5.3.2.7.2	Local Bus Debug Configuration	5-28
5.3.2.8	DDR Control Driver Register (DDRCDR)	5-28
5.3.2.9	DDR Debug Status Register (DDRDSR)	5-30
5.4	Software Watchdog Timer (WDT)	5-31
5.4.1	Overview	5-31
5.4.2	Features	5-32
5.4.3	Modes of Operation	5-32
5.4.4	Memory Map/Register Definition	5-33
5.4.4.1	System Watchdog Control Register (SWCRR)	5-33
5.4.4.2	System Watchdog Count Register (SWCNR)	5-34
5.4.4.3	System Watchdog Service Register (SWSRR)	5-34
5.4.5	Functional Description	5-35
5.4.5.1	Software Watchdog Timer Unit	5-35
5.4.5.2	Modes of Operation	5-37
5.4.6	Initialization/Application Information	5-38
5.4.6.1	WDT Programming Guidelines	5-38
5.5	Real Time Clock Module (RTC)	5-38
5.5.1	Overview	5-38
5.5.2	Features	5-39
5.5.3	Modes of Operation	5-39
5.5.4	External Signal Description	5-39
5.5.4.1	Overview	5-39
5.5.4.2	Detailed Signal Descriptions	5-39
5.5.5	Memory Map/Register Definition	5-40
5.5.5.1	Real Time Counter Control Register (RTCNR)	5-40
5.5.5.2	Real Time Counter Load Register (RTLDR)	5-41
5.5.5.3	Real Time Counter Prescale Register (RTPSR)	5-41
5.5.5.4	Real Time Counter Register (RTCTR)	5-42
5.5.5.5	Real Time Counter Event Register (RTEVR)	5-42
5.5.5.6	Real Time Counter Alarm Register (RTALR)	5-43
5.5.6	Functional Description	5-43

Contents

Paragraph Number	Title	Page Number
5.5.6.1	Real Time Counter Unit	5-43
5.5.6.2	RTC Operational Modes	5-44
5.5.7	RTC Programming Guidelines	5-45
5.6	Periodic Interval Timer (PIT)	5-45
5.6.1	Overview	5-45
5.6.2	Features	5-46
5.6.3	Modes of Operation	5-46
5.6.4	External Signal Description	5-46
5.6.4.1	Overview	5-46
5.6.4.2	Detailed Signal Description	5-47
5.6.5	Memory Map/Register Definition	5-47
5.6.5.1	Periodic Interval Timer Control Register (PTCNR)	5-47
5.6.5.2	Periodic Interval Timer Load Register (PTLDR)	5-48
5.6.5.3	Periodic Interval Timer Prescale Register (PTPSR)	5-49
5.6.5.4	Periodic Interval Timer Counter Register (PTCTR)	5-49
5.6.5.5	Periodic Interval Timer Event Register (PTEVR)	5-50
5.6.6	Functional Description	5-50
5.6.6.1	Periodic Interval Timer Unit	5-50
5.6.6.2	PIT Operational Modes	5-51
5.6.7	PIT Programming Guidelines	5-51
5.7	General-Purpose Timers (GTMs)	5-52
5.7.1	Overview	5-52
5.7.2	Features	5-53
5.7.3	Modes of Operation	5-53
5.7.3.1	Cascaded Modes	5-53
5.7.3.2	Clock Source Modes	5-54
5.7.3.3	Reference Modes	5-54
5.7.3.4	Capture Modes	5-54
5.7.4	External Signal Description	5-54
5.7.4.1	Overview	5-54
5.7.4.2	Detailed Signal Descriptions	5-55
5.7.5	Memory Map/Register Definition	5-56
5.7.5.1	Global Timers Configuration Registers (GTCFR*n*)	5-57
5.7.5.2	Global Timers Mode Registers (GTMDR1–GTMDR4)	5-60
5.7.5.3	Global Timers Reference Registers (GTRFR1–GTRFR4)	5-62
5.7.5.4	Global Timers Capture Registers (GTCPR1–GTCPR4)	5-62
5.7.5.5	Global Timers Counter Registers (GTCNR1–GTCNR4)	5-63
5.7.5.6	Global Timers Event Registers (GTEVR1–GTEVR4)	5-63
5.7.5.7	Global Timers Prescale Registers (GTPSR1–GTPSR4)	5-64
5.7.6	Functional Description	5-65
5.7.6.1	General-Purpose Timer Units	5-65

Contents

Paragraph Number	Title	Page Number
5.7.6.2	Reference Modes	5-65
5.7.6.3	Capture Modes	5-65
5.7.6.4	Cascaded Modes	5-66
5.7.7	Initialization/Application Information	5-68
5.7.7.1	Programming Guidelines	5-68
5.7.7.1.1	GTM Registers	5-68
5.8	Power Management Control	5-68
5.8.1	Modes of Operation	5-68
5.8.2	External Signal Description	5-69
5.8.3	Memory Map/Register Definition	5-69
5.8.3.1	Power Management Controller Configuration Register (PMCCR)	5-69
5.8.3.2	Power Management Controller Event Register (PMCER)	5-70
5.8.3.3	Power Management Controller Mask Register (PMCMR)	5-71
5.8.4	Functional Description	5-71
5.8.4.1	Dynamic Power Management	5-71
5.8.4.2	Shutting Down Unused Blocks	5-72
5.8.4.3	Software-Controlled Power-Down States	5-72
5.8.4.3.1	Entering Low Power States—Core-Only Mode	5-72
5.8.4.3.2	Entering Low Power States—Core and System Mode	5-72
5.8.4.4	Exiting Core and System Low Power States	5-73
5.8.4.4.1	Exiting Low Power States—Core-Only Mode	5-73
5.8.4.4.2	Exiting Low Power States—Core and System Mode	5-73
5.8.5	Initialization/Application Information	5-74
5.8.5.1	Core Disable in Low Power Mode	5-74

Chapter 6
Arbiter and Bus Monitor

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)	6-2
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)	6-7
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

Contents

Paragraph Number	Title	Page Number
6.3.1.1	Address Bus Arbitration with PRIORITY[0:1]	6-12
6.3.1.2	Address Bus Arbitration with REPEAT	6-13
6.3.1.3	Address Bus Arbitration after ARTRY	6-13
6.3.1.4	Address Bus Parking	6-13
6.3.1.5	Data Bus Arbitration	6-13
6.3.2	Bus Error Detection	6-14
6.3.2.1	Address Time Out	6-14
6.3.2.2	Data Time Out	6-14
6.3.2.3	Transfer Error	6-14
6.3.2.4	Address Only Transaction Type	6-15
6.3.2.5	Reserved Transaction Type	6-15
6.3.2.6	Illegal (eciwx/ecowx) Transaction Type	6-16
6.4	Initialization/Applications Information	6-16
6.4.1	Initialization Sequence	6-16
6.4.2	Error Handling Sequence	6-17

Chapter 7
e300 Processor Core Overview

7.1	Overview	7-1
7.1.1	Features	7-3
7.1.2	Instruction Unit	7-6
7.1.2.1	Instruction Queue and Dispatch Unit	7-6
7.1.2.2	Branch Processing Unit (BPU)	7-7
7.1.3	Independent Execution Units	7-7
7.1.3.1	Integer Unit (IU)	7-7
7.1.3.2	Floating-Point Unit (FPU)	7-7
7.1.3.3	Load/Store Unit (LSU)	7-8
7.1.3.4	System Register Unit (SRU)	7-8
7.1.4	Completion Unit	7-8
7.1.5	Memory Subsystem Support	7-8
7.1.5.1	Memory Management Units (MMUs)	7-9
7.1.5.2	Cache Units	7-10
7.1.6	Bus Interface Unit (BIU)	7-10
7.1.7	System Support Functions	7-11
7.1.7.1	Power Management	7-11
7.1.7.2	Time Base/Decrementer	7-11
7.1.7.3	JTAG Test and Debug Interface	7-12
7.1.7.4	Clock Multiplier	7-12
7.2	PowerPC Architecture Implementation	7-12
7.3	Implementation-Specific Information	7-12

Contents

Paragraph Number	Title	Page Number
7.3.1	Register Model	7-13
7.3.1.1	UISA Registers	7-15
7.3.1.1.1	General-Purpose Registers (GPRs)	7-15
7.3.1.1.2	Floating-Point Registers (FPRs)	7-15
7.3.1.1.3	Condition Register (CR)	7-15
7.3.1.1.4	Floating-Point Status and Control Register (FPSCR)	7-15
7.3.1.1.5	User-Level SPRs	7-15
7.3.1.2	VEA Registers	7-16
7.3.1.3	OEA Registers	7-16
7.3.1.3.1	Machine State Register (MSR)	7-16
7.3.1.3.2	Segment Registers (SRs)	7-18
7.3.1.3.3	Supervisor-Level SPRs	7-18
7.3.2	Instruction Set and Addressing Modes	7-25
7.3.2.1	PowerPC Instruction Set and Addressing Modes	7-25
7.3.2.2	Implementation-Specific Instruction Set	7-26
7.3.3	Cache Implementation	7-27
7.3.3.1	PowerPC Cache Characteristics	7-27
7.3.3.2	Implementation-Specific Cache Implementation	7-27
7.3.3.3	Instruction and Data Cache Way-Locking	7-29
7.3.4	Interrupt Model	7-29
7.3.4.1	PowerPC Interrupt Model	7-29
7.3.4.2	Implementation-Specific Interrupt Model	7-30
7.3.5	Memory Management	7-33
7.3.5.1	PowerPC Memory Management	7-33
7.3.5.2	Implementation-Specific Memory Management	7-33
7.3.6	Instruction Timing	7-34
7.3.7	Core Interface	7-35
7.3.7.1	Memory Accesses	7-36
7.3.7.2	Signals	7-36
7.3.8	Debug Features	7-37
7.3.8.1	Breakpoint Signaling	7-37
7.4	Differences Between Cores	7-38

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	8-5

Contents

Paragraph Number	Title	Page Number
8.4	External Signal Description	8-5
8.4.1	Overview	8-5
8.4.2	Detailed Signal Descriptions	8-6
8.5	Memory Map/Register Definition	8-6
8.5.1	System Global Interrupt Configuration Register (SICFR)	8-8
8.5.2	System Global Interrupt Vector Register (SIVCR)	8-9
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)	8-14
8.5.5	System Internal Interrupt Group D Priority Register (SIPRR_D)	8-14
8.5.6	System Internal Interrupt Mask Register (SIMSR_H and SIMSR_L)	8-15
8.5.7	System Internal Interrupt Control Register (SICNR)	8-16
8.5.8	System External Interrupt Pending Register (SEPNR)	8-18
8.5.9	System Mixed Interrupt Group A Priority Register (SMPRR_A)	8-18
8.5.10	System Mixed Interrupt Group B Priority Register (SMPRR_B)	8-19
8.5.11	System External Interrupt Mask Register (SEMSR)	8-20
8.5.12	System External Interrupt Control Register (SECNR)	8-21
8.5.13	System Error Status Register (SERSR)	8-22
8.5.14	System Error Mask Register (SERMR)	8-23
8.5.15	System Error Control Register (SERCR)	8-24
8.5.16	System Internal Interrupt Force Registers (SIFCR_H and SIFCR_L)	8-25
8.5.17	System External Interrupt Force Register (SEFCR)	8-26
8.5.18	System Error Force Register (SERFR)	8-26
8.5.19	System Critical Interrupt Vector Register (SCVCR)	8-27
8.5.20	System Management Interrupt Vector Register (SMVCR)	8-27
8.6	Functional Description	8-28
8.6.1	Interrupt Types	8-28
8.6.2	Interrupt Configuration	8-29
8.6.3	Internal Interrupts Group Relative Priority	8-30
8.6.4	Mixed Interrupts Group Relative Priority	8-30
8.6.5	Highest Priority Interrupt	8-31
8.6.6	Interrupt Source Priorities	8-31
8.6.7	Masking Interrupt Sources	8-34
8.6.8	Interrupt Vector Generation and Calculation	8-35
8.6.9	Machine Check Interrupts	8-35

Chapter 9
DDR Memory Controller

9.1	Introduction	9-1
9.2	Features	9-2
9.2.1	Modes of Operation	9-3

Contents

Paragraph Number	Title	Page Number
9.3	External Signal Descriptions	9-3
9.3.1	Signals Overview	9-3
9.3.2	Detailed Signal Descriptions	9-5
9.3.2.1	Memory Interface Signals	9-5
9.3.2.2	Clock Interface Signals	9-8
9.3.2.3	Debug Signals	9-9
9.4	Memory Map/Register Definition	9-9
9.4.1	Register Descriptions	9-10
9.4.1.1	Chip Select Memory Bounds (CS*n*_BNDS)	9-10
9.4.1.2	Chip Select Configuration (CS*n*_CONFIG)	9-11
9.4.1.3	DDR SDRAM Timing Configuration 3 (TIMING_CFG_3)	9-13
9.4.1.4	DDR SDRAM Timing Configuration 0 (TIMING_CFG_0)	9-14
9.4.1.5	DDR SDRAM Timing Configuration 1 (TIMING_CFG_1)	9-16
9.4.1.6	DDR SDRAM Timing Configuration 2 (TIMING_CFG_2)	9-18
9.4.1.7	DDR SDRAM Control Configuration (DDR_SDRAM_CFG)	9-20
9.4.1.8	DDR SDRAM Control Configuration 2 (DDR_SDRAM_CFG_2)	9-22
9.4.1.9	DDR SDRAM Mode Configuration (DDR_SDRAM_MODE)	9-24
9.4.1.10	DDR SDRAM Mode 2 Configuration (DDR_SDRAM_MODE_2)	9-24
9.4.1.11	DDR SDRAM Mode Control Register (DDR_SDRAM_MD_CNTL)	9-25
9.4.1.12	DDR SDRAM Interval Configuration (DDR_SDRAM_INTERVAL)	9-27
9.4.1.13	DDR SDRAM Data Initialization (DDR_DATA_INIT)	9-28
9.4.1.14	DDR SDRAM Clock Control (DDR_SDRAM_CLK_CNTL)	9-28
9.4.1.15	DDR Initialization Address (DDR_INIT_ADDR)	9-29
9.4.1.16	DDR IP Block Revision 1 (DDR_IP_REV1)	9-30
9.4.1.17	DDR IP Block Revision 2 (DDR_IP_REV2)	9-30
9.4.1.18	Memory Data Path Error Injection Mask High (DATA_ERR_INJECT_HI)	9-31
9.4.1.19	Memory Data Path Error Injection Mask Low (DATA_ERR_INJECT_LO)	9-31
9.4.1.20	Memory Data Path Error Injection Mask ECC (ERR_INJECT)	9-32
9.4.1.21	Memory Data Path Read Capture High (CAPTURE_DATA_HI)	9-32
9.4.1.22	Memory Data Path Read Capture Low (CAPTURE_DATA_LO)	9-33
9.4.1.23	Memory Data Path Read Capture ECC (CAPTURE_ECC)	9-33
9.4.1.24	Memory Error Detect (ERR_DETECT)	9-33
9.4.1.25	Memory Error Disable (ERR_DISABLE)	9-34
9.4.1.26	Memory Error Interrupt Enable (ERR_INT_EN)	9-35
9.4.1.27	Memory Error Attributes Capture (CAPTURE_ATTRIBUTES)	9-36
9.4.1.28	Memory Error Address Capture (CAPTURE_ADDRESS)	9-37
9.4.1.29	Single-Bit ECC Memory Error Management (ERR_SBE)	9-37
9.5	Functional Description	9-38
9.5.1	DDR SDRAM Interface Operation	9-43
9.5.1.1	Supported DDR SDRAM Organizations	9-43
9.5.2	DDR SDRAM Address Multiplexing	9-45

Contents

Paragraph Number	Title	Page Number
9.5.3	JEDEC Standard DDR SDRAM Interface Commands	9-49
9.5.4	DDR SDRAM Interface Timing	9-51
9.5.4.1	Clock Distribution	9-54
9.5.5	DDR SDRAM Mode-Set Command Timing	9-55
9.5.6	DDR SDRAM Registered DIMM Mode	9-56
9.5.7	DDR SDRAM Write Timing Adjustments	9-56
9.5.8	DDR SDRAM Refresh	9-57
9.5.8.1	DDR SDRAM Refresh Timing	9-58
9.5.8.2	DDR SDRAM Refresh and Power-Saving Modes	9-58
9.5.8.2.1	Self-Refresh in Sleep Mode	9-60
9.5.9	DDR Data Beat Ordering	9-61
9.5.10	Page Mode and Logical Bank Retention	9-61
9.5.11	Error Checking and Correcting (ECC)	9-62
9.5.12	Error Management	9-64
9.6	Initialization/Application Information	9-65
9.6.1	Programming Differences Between Memory Types	9-66
9.6.2	DDR SDRAM Initialization Sequence	9-69
9.6.3	Using Forced Self-Refresh Mode to Implement a Battery-Backed RAM System	9-69
9.6.3.1	Hardware Based Self-Refresh Scheme	9-69
9.6.3.2	Software Based Self-Refresh Scheme	9-70
9.6.3.3	Bypassing Re-initialization During Battery-Backed Operation	9-70

Chapter 10
Local Bus Controller

10.1	Introduction	10-1
10.1.1	Features	10-2
10.1.2	Modes of Operation	10-3
10.1.2.1	LBC Bus Clock and Clock Ratios	10-3
10.1.2.2	Source ID Debug Mode	10-3
10.2	External Signal Descriptions	10-4
10.3	Memory Map/Register Definition	10-9
10.3.1	Register Descriptions	10-10
10.3.1.1	Base Registers (BR0–BR7)	10-11
10.3.1.2	Option Registers (OR0–OR7)	10-12
10.3.1.2.1	Address Mask	10-12
10.3.1.2.2	Option Registers (OR*n*)—GPCM Mode	10-13
10.3.1.2.3	Option Registers (OR*n*)—UPM Mode	10-16
10.3.1.2.4	Option Registers (OR*n*)—SDRAM Mode	10-17
10.3.1.3	UPM Memory Address Register (MAR)	10-18

Contents

Paragraph Number	Title	Page Number
10.3.1.4	UPM Mode Registers (M*n*MR)	10-19
10.3.1.5	Memory Refresh Timer Prescaler Register (MRTPR)	10-21
10.3.1.6	UPM Data Register (MDR)	10-22
10.3.1.7	Local Bus SDRAM Machine Mode Register (LSDMR)	10-22
10.3.1.8	UPM Refresh Timer (LURT)	10-24
10.3.1.9	SDRAM Refresh Timer (LSRT)	10-25
10.3.1.10	Transfer Error Status Register (LTESR)	10-26
10.3.1.11	Transfer Error Check Disable Register (LTEDR)	10-27
10.3.1.12	Transfer Error Interrupt Enable Register (LTEIR)	10-27
10.3.1.13	Transfer Error Attributes Register (LTEATR)	10-28
10.3.1.14	Transfer Error Address Register (LTEAR)	10-29
10.3.1.15	Local Bus Configuration Register (LBCR)	10-29
10.3.1.16	Clock Ratio Register (LCRR)	10-30
10.4	Functional Description	10-31
10.4.1	Basic Architecture	10-32
10.4.1.1	Address and Address Space Checking	10-32
10.4.1.2	External Address Latch Enable Signal (LALE)	10-33
10.4.1.3	Data Transfer Acknowledge (TA)	10-34
10.4.1.4	Data Buffer Control (LBCTL)	10-35
10.4.1.5	Parity Generation and Checking (LDP)	10-35
10.4.1.6	Bus Monitor	10-35
10.4.2	General-Purpose Chip-Select Machine (GPCM)	10-35
10.4.2.1	Timing Configuration	10-37
10.4.2.2	Chip-Select Assertion Timing	10-40
10.4.2.2.1	Programmable Wait State Configuration	10-41
10.4.2.2.2	Chip-Select and Write Enable Negation Timing	10-41
10.4.2.2.3	Relaxed Timing	10-42
10.4.2.2.4	Output Enable (LOE) Timing	10-45
10.4.2.2.5	Extended Hold Time on Read Accesses	10-45
10.4.2.3	External Access Termination (LGTA)	10-46
10.4.2.4	Boot Chip-Select Operation	10-47
10.4.3	SDRAM Machine	10-48
10.4.3.1	Supported SDRAM Configurations	10-48
10.4.3.2	SDRAM Power-On Initialization	10-49
10.4.3.3	Intel PC133 and JEDEC-Standard SDRAM Interface Commands	10-50
10.4.3.4	Page Hit Checking	10-51
10.4.3.5	Page Management	10-51
10.4.3.6	SDRAM Address Multiplexing	10-51
10.4.3.7	SDRAM Device-Specific Parameters	10-52
10.4.3.7.1	Precharge-to-Activate Interval	10-52
10.4.3.7.2	Activate-to-Read/Write Interval	10-53

Contents

Paragraph Number	Title	Page Number
10.4.3.7.3	Column Address to First Data Out—CAS Latency	10-53
10.4.3.7.4	Last Data In to Precharge—Write Recovery	10-54
10.4.3.7.5	Refresh Recovery Interval (RFRC)	10-54
10.4.3.7.6	External Address and Command Buffers (BUFCMD)	10-55
10.4.3.8	SDRAM Interface Timing	10-55
10.4.3.9	SDRAM Read/Write Transactions	10-57
10.4.3.10	SDRAM MODE-SET Command Timing	10-58
10.4.3.11	SDRAM Refresh	10-58
10.4.3.11.1	SDRAM Refresh Timing	10-58
10.4.4	User-Programmable Machines (UPMs)	10-59
10.4.4.1	UPM Requests	10-60
10.4.4.1.1	Memory Access Requests	10-61
10.4.4.1.2	UPM Refresh Timer Requests	10-61
10.4.4.1.3	Software Requests—RUN Command	10-62
10.4.4.1.4	Exception Requests	10-62
10.4.4.2	Programming the UPMs	10-62
10.4.4.2.1	UPM Programming Example (Two Sequential Writes to the RAM Array)	10-63
10.4.4.2.2	UPM Programming Example (Two Sequential Reads from the RAM Array)	10-64
10.4.4.3	UPM Signal Timing	10-64
10.4.4.4	UPM RAM Array	10-65
10.4.4.4.1	UPM RAM Words	10-66
10.4.4.4.2	Chip-Select Signal Timing (CST*n*)	10-69
10.4.4.4.3	Byte Select Signal Timing (BST*n*)	10-69
10.4.4.4.4	General-Purpose Signals (G*n*T*n*, GO*n*)	10-70
10.4.4.4.5	Loop Control (LOOP)	10-70
10.4.4.4.6	Repeat Execution of Current RAM Word (REDO)	10-71
10.4.4.4.7	Address Multiplexing (AMX)	10-71
10.4.4.4.8	Data Valid and Data Sample Control (UTA)	10-72
10.4.4.4.9	LGPL[0:5] Signal Negation (LAST)	10-73
10.4.4.4.10	Wait Mechanism (WAEN)	10-73
10.4.4.5	Synchronous Sampling of LUPWAIT for Early Transfer Acknowledge	10-74
10.4.4.6	Extended Hold Time on Read Accesses	10-74
10.4.4.7	Memory System Interface Example Using UPM	10-74
10.5	Initialization/Application Information	10-80
10.5.1	Interfacing to Peripherals in Multiplexed Address/Data Mode	10-80
10.5.1.1	Multiplexed Address/Data Bus and Unmultiplexed Address Signals	10-80
10.5.1.2	Peripheral Hierarchy on the Local Bus	10-81
10.5.1.3	Peripheral Hierarchy on the Local Bus for Very High Bus Speeds	10-81
10.5.1.4	GPCM Timings	10-82

Contents

Paragraph Number	Title	Page Number
10.5.2	Bus Turnaround	10-83
10.5.2.1	Address Phase after Previous Read	10-83
10.5.2.2	Read Data Phase after Address Phase	10-83
10.5.2.3	Read-Modify-Write Cycle for Parity Protected Memory Banks	10-84
10.5.2.4	UPM Cycles with Additional Address Phases	10-84
10.5.3	Interface to Different Port-Size Devices	10-84
10.5.4	Interfacing to SDRAM	10-86
10.5.4.1	Basic SDRAM Capabilities of the Local Bus	10-86
10.5.4.2	Maximum Amount of SDRAM Supported	10-87
10.5.4.3	SDRAM Machine Limitations	10-88
10.5.4.3.1	Analysis of Maximum Row Number Due to Bank Select Multiplexing	10-88
10.5.4.3.2	Bank Select Signals	10-88
10.5.4.3.3	128-Mbyte SDRAM	10-89
10.5.4.3.4	256-Mbyte SDRAM	10-91
10.5.4.3.5	512-Mbyte SDRAM	10-91
10.5.4.4	Parity Support for SDRAM	10-93
10.5.5	Interfacing to ZBT SRAM	10-94
10.5.6	Interfacing to DSP Host Ports	10-95
10.5.6.1	Interfacing to MSC8122 DSI	10-95
10.5.6.2	DSI in Asynchronous SRAM-Like Mode	10-96
10.5.6.3	DSI in Synchronous Mode	10-98
10.5.6.3.1	Synchronous Single Write	10-101
10.5.6.3.2	Synchronous Single Read	10-102
10.5.6.3.3	Synchronous Burst Write	10-103
10.5.6.3.4	Synchronous Burst Read	10-104
10.5.6.4	Broadcast Accesses	10-104

Chapter 11
Sequencer

11.1	Overview	11-1
11.1.1	Features	11-2
11.2	External Signal Description	11-2
11.3	Memory Map/Register Definition	11-2
11.4	Register Descriptions	11-3
11.4.1	PCI Outbound Translation Address Registers (POTARn)	11-3
11.4.2	PCI Outbound Base Address Registers (POBARn)	11-3
11.4.3	PCI Outbound Comparison Mask Registers (POCMRn)	11-4
11.4.4	Power Management Control Register (PMCR)	11-5
11.4.5	Discard Timer Control Register (DTCR)	11-6

Contents

Paragraph Number	Title	Page Number
11.5	Functional Description	11-6
11.5.1	Transaction Forwarding	11-6
11.5.1.1	Transactions from the Coherency System Bus (CSB) Port	11-7
11.5.1.2	Transactions from the PCI Ports	11-7
11.5.1.3	Transactions from the DMA Port	11-7
11.5.2	PCI Outbound Address Translation	11-7
11.5.3	Transaction Ordering	11-8

Chapter 12
DMA/Messaging Unit

12.1	Features	12-2
12.2	External Signal Description	12-2
12.2.1	Detailed Signal Descriptions	12-2
12.3	Memory Map/Register Definition	12-3
12.4	Register Descriptions	12-4
12.4.1	Outbound Message Interrupt Status Register (OMISR)	12-4
12.4.2	Outbound Message Interrupt Mask Register (OMIMR)	12-6
12.4.3	Inbound Message Registers	12-7
12.4.4	Outbound Message Registers (OMR0–OMR1)	12-7
12.4.5	Doorbell Registers	12-8
12.4.5.1	Outbound Doorbell Register (ODR)	12-8
12.4.5.2	Inbound Doorbell Register (IDR)	12-9
12.4.6	Inbound Message Interrupt Status Register (IMISR)	12-9
12.4.7	Inbound Message Interrupt Mask Register (IMIMR)	12-11
12.4.8	DMA Registers	12-11
12.4.8.1	DMA Mode Register (DMAMR*n*)	12-12
12.4.8.2	DMA Status Register (DMASR*n*)	12-14
12.4.8.3	DMA Current Descriptor Address Register (DMACDAR*n*)	12-15
12.4.8.4	DMA Source Address Register (DMASAR*n*)	12-16
12.4.8.5	DMA Destination Address Register (DMADAR*n*)	12-16
12.4.8.6	DMA Byte Count Register (DMABCR*n*)	12-17
12.4.8.7	DMA Next Descriptor Address Register (DMANDAR*n*)	12-17
12.4.8.8	DMA General Status Register (DMAGSR)	12-18
12.5	Functional Description	12-18
12.5.1	Message Unit	12-18
12.5.1.1	Messaging Registers (IMR0–IMR1)	12-19
12.5.1.2	Doorbell Registers (IDR and ODR)	12-19
12.5.2	DMA Controller	12-19
12.5.3	DMA Operation	12-20
12.5.3.1	External Control	12-21

Contents

Paragraph Number	Title	Page Number
12.5.3.2	DMA Coherency	12-22
12.5.3.3	Halt and Error Conditions	12-22
12.5.4	DMA Segment Descriptors	12-22
12.5.4.1	Descriptor in Big-Endian Mode	12-23
12.5.4.2	Descriptor in Little-Endian Mode	12-24
12.6	Initialization/Application Information	12-24
12.6.1	Initialization Steps in Direct Mode	12-24
12.6.2	Initialization Steps in Chaining Mode	12-24
12.6.3	Initialization Steps in Direct Mode with External Control	12-25
12.6.4	Initialization Steps in Chaining Mode with External Control	12-25

Chapter 13
PCI Bus Interface

13.1	Introduction	13-2
13.1.1	Features	13-3
13.1.2	Modes of Operation	13-3
13.1.2.1	Host/Agent Mode Configuration	13-4
13.1.2.2	Single 64-Bit/Dual 32-Bit Interface Configuration	13-4
13.1.2.3	PCI Arbiter Configuration	13-4
13.2	External Signal Description	13-4
13.3	Memory Map/Register Definitions	13-14
13.3.1	PCI Configuration Access Registers	13-16
13.3.1.1	PCI_CONFIG_ADDRESS	13-16
13.3.1.2	PCI_CONFIG_DATA	13-18
13.3.1.3	PCI Interrupt Acknowledge Register (PCI_INT_ACK)	13-18
13.3.2	PCI Memory-Mapped Control and Status Registers	13-18
13.3.2.1	PCI Error Status Register (PCI_ESR)	13-18
13.3.2.2	PCI Error Capture Disable Register (PCI_ECDR)	13-19
13.3.2.3	PCI Error Enable Register (PCI_EER)	13-20
13.3.2.4	PCI Error Attributes Capture Register (PCI_EATCR)	13-21
13.3.2.5	PCI Error Address Capture Register (PCI_EACR)	13-23
13.3.2.6	PCI Error Extended Address Capture Register (PCI_EEACR)	13-23
13.3.2.7	PCI Error Data Low Capture Register (PCI_EDLCR)	13-24
13.3.2.8	PCI Error Data High Capture Register (PCI_EDHCR)	13-24
13.3.2.9	PCI General Control Register (PCI_GCR)	13-24
13.3.2.10	PCI Error Control Register (PCI_ECR)	13-25
13.3.2.11	PCI General Status Register (PCI_GSR)	13-26
13.3.2.12	PCI Inbound Translation Address Registers (PITARn)	13-26
13.3.2.13	PCI Inbound Base Address Registers (PIBARn)	13-27
13.3.2.14	PCI Inbound Extended Base Address Registers (PIEBARn)	13-27

Contents

Paragraph Number	Title	Page Number
13.3.2.15	PCI Inbound Window Attribute Registers (PIWARn)	13-28
13.3.3	PCI Configuration Space Registers	13-29
13.3.3.1	Vendor ID Configuration Register	13-31
13.3.3.2	Device ID Configuration Register	13-31
13.3.3.3	PCI Command Configuration Register	13-32
13.3.3.4	PCI Status Configuration Register	13-33
13.3.3.5	Revision ID Configuration Register	13-34
13.3.3.6	Standard Programming Interface Configuration Register	13-34
13.3.3.7	Subclass Code Configuration Register	13-35
13.3.3.8	Base Class Code Configuration Register	13-35
13.3.3.9	Cache Line Size Configuration Register	13-36
13.3.3.10	Latency Timer Configuration Register	13-36
13.3.3.11	Header Type Configuration Register	13-37
13.3.3.12	BIST Control Configuration Register	13-37
13.3.3.13	PIMMR Base Address Configuration Register	13-37
13.3.3.14	GPL Base Address Register 0	13-38
13.3.3.15	GPL Base Address Registers 1–2	13-38
13.3.3.16	GPL Extended Base Address Registers 1–2	13-39
13.3.3.17	Subsystem Vendor ID Configuration Register	13-39
13.3.3.18	Subsystem Device ID Configuration Register	13-40
13.3.3.19	Capabilities Pointer Configuration Register	13-40
13.3.3.20	Interrupt Line Configuration Register	13-41
13.3.3.21	Interrupt Pin Configuration Register	13-41
13.3.3.22	Minimum Grant Configuration Register	13-41
13.3.3.23	Maximum Latency Configuration Register	13-42
13.3.3.24	PCI Function Configuration Register	13-42
13.3.3.25	PCI Arbiter Control Register (PCIACR)	13-43
13.3.3.26	Hot Swap Register Block	13-44
13.4	Functional Description	13-45
13.4.1	PCI Bus Arbitration	13-45
13.4.1.1	Bus Parking	13-45
13.4.1.2	Arbitration Algorithm	13-45
13.4.1.3	Broken Master Lock-Out	13-46
13.4.1.4	Master Latency Timer	13-47
13.4.2	Bus Commands	13-47
13.4.3	PCI Protocol Fundamentals	13-48
13.4.3.1	Basic Transfer Control	13-48
13.4.3.2	Addressing	13-48
13.4.3.3	Device Selection	13-49
13.4.3.4	Byte Enable Signals	13-49
13.4.3.5	Bus Driving and Turnaround	13-50

Contents

Paragraph Number	Title	Page Number
13.4.3.6	Bus Transactions	13-50
13.4.3.7	Read and Write Transactions	13-50
13.4.3.8	Transaction Termination	13-52
13.4.4	Other Bus Operations	13-55
13.4.4.1	Fast Back-to-Back Transactions	13-55
13.4.4.2	Dual Address Cycles	13-55
13.4.4.3	Data Streaming	13-55
13.4.4.4	Host Mode Configuration Access	13-56
13.4.4.5	Agent Mode Configuration Access	13-57
13.4.4.6	Special Cycle Command	13-57
13.4.4.7	Interrupt Acknowledge	13-58
13.4.5	Error Functions	13-58
13.4.5.1	Parity	13-58
13.4.5.2	Error Reporting	13-58
13.4.6	PCI Inbound Address Translation	13-60
13.4.7	CompactPCI Hot Swap Specification Support	13-61
13.5	Initialization/Application Information	13-61
13.5.1	Initialization Sequence for Host Mode	13-61
13.5.2	Initialization Sequence for Agent Mode	13-61

Chapter 14
Security Engine (SEC) 2.4

14.1	Overview	14-1
14.2	Architecture Overview	14-2
14.2.1	Data Packet Descriptors	14-4
14.2.2	Execution Units (EUs)	14-5
14.2.2.1	Public Key Execution Unit (PKEU)	14-5
14.2.2.1.1	Elliptic Curve Operations	14-5
14.2.2.1.2	Modular Exponentiation Operations	14-6
14.2.2.2	Data Encryption Standard Execution Unit (DEU)	14-6
14.2.2.3	ARC Four Execution Unit (AFEU)	14-6
14.2.2.4	Advanced Encryption Standard Execution Unit (AESU)	14-7
14.2.2.5	Message Digest Execution Unit (MDEU)	14-7
14.2.2.6	Random Number Generator (RNG)	14-7
14.2.3	Crypto-Channels	14-8
14.2.4	SEC Controller	14-9
14.2.4.1	Host-Controlled Access	14-9
14.2.4.2	Channel-Controlled Access	14-9
14.2.5	Bus Interface	14-10
14.3	Configuration of Internal Memory Space	14-10

Contents

Paragraph Number	Title	Page Number
14.4	Descriptor Overview	14-15
14.4.1	Descriptor Structure	14-15
14.4.2	Descriptor Format—Header Dword	14-16
14.4.2.1	Selecting Execution Units—EU_SEL0 and EU_SEL1	14-17
14.4.2.2	Selecting Descriptor Type—DESC_TYPE	14-18
14.4.3	Descriptor Format—Pointer Dwords	14-19
14.4.4	Link Table Format	14-21
14.4.4.1	Link Table Example	14-23
14.4.5	Descriptor Types	14-24
14.5	Execution Units	14-26
14.5.1	Public Key Execution Unit (PKEU)	14-26
14.5.1.1	PKEU Mode Register (PKEUMR)	14-26
14.5.1.2	PKEU Key Size Register (PKEUKSR)	14-28
14.5.1.3	PKEU AB Size Register (PKEUABS)	14-28
14.5.1.4	PKEU Data Size Register (PKEUDSR)	14-29
14.5.1.5	PKEU Reset Control Register (PKEURCR)	14-29
14.5.1.6	PKEU Status Register (PKEUSR)	14-30
14.5.1.7	PKEU Interrupt Status Register (PKEUISR)	14-31
14.5.1.8	PKEU Interrupt Control Register (PKEUICR)	14-32
14.5.1.9	PKEU EU-Go Register (PKEUEUG)	14-33
14.5.1.10	PKEU Parameter Memories	14-34
14.5.1.10.1	PKEU Parameter Memory A	14-34
14.5.1.10.2	PKEU Parameter Memory B	14-34
14.5.1.10.3	PKEU Parameter Memory E	14-34
14.5.1.10.4	PKEU Parameter Memory N	14-34
14.5.2	Data Encryption Standard Execution Unit (DEU)	14-34
14.5.2.1	DEU Mode Register (DEUMR)	14-35
14.5.2.2	DEU Key Size Register (DEUKSR)	14-36
14.5.2.3	DEU Data Size Register (DEUDSR)	14-36
14.5.2.4	DEU Reset Control Register (DEURCR)	14-37
14.5.2.5	DEU Status Register (DEUSR)	14-37
14.5.2.6	DEU Interrupt Status Register (DEUISR)	14-38
14.5.2.7	DEU Interrupt Control Register (DEUICR)	14-40
14.5.2.8	DEU EU-Go Register (DEUEUG)	14-41
14.5.2.9	DEU IV Register (DEUIV)	14-42
14.5.2.10	DEU Key Registers (DEUK1–DEUK3)	14-42
14.5.2.11	DEU FIFOs	14-42
14.5.3	ARC Four Execution Unit (AFEU)	14-42
14.5.3.1	AFEU Mode Register (AFEUMR)	14-43
14.5.3.2	Host-Provided Context through Prevent Permute	14-43
14.5.3.2.1	Dump Context	14-43

Contents

Paragraph Number	Title	Page Number
14.5.3.3	AFEU Key Size Register (AFEUKSR)	14-44
14.5.3.4	AFEU Context/Data Size Register (AFEUDSR)	14-45
14.5.3.5	AFEU Reset Control Register (AFEURCR)	14-46
14.5.3.6	AFEU Status Register (AFEUSR)	14-46
14.5.3.7	AFEU Interrupt Status Register (AFEUISR)	14-47
14.5.3.8	AFEU Interrupt Control Register (AFEUICR)	14-49
14.5.3.9	AFEU End of Message Register (AFEUEMR)	14-50
14.5.3.10	AFEU Context	14-50
14.5.3.10.1	AFEU Context Memory	14-50
14.5.3.10.2	AFEU Context Memory Pointer Register	14-51
14.5.3.11	AFEU Key Registers (AFEUK0, AFEUK1)	14-51
14.5.3.11.1	AFEU FIFOs	14-51
14.5.4	Message Digest Execution Unit (MDEU)	14-51
14.5.4.1	MDEU Mode Register (MDEUMR)	14-51
14.5.4.2	Recommended Settings for MDEUMR	14-54
14.5.4.3	MDEU Key Size Register (MDEUKSR)	14-55
14.5.4.4	MDEU Data Size Register (MDEUDSR)	14-56
14.5.4.5	MDEU Reset Control Register (MDEURCR)	14-56
14.5.4.6	MDEU Status Register (MDEUSR)	14-57
14.5.4.7	MDEU Interrupt Status Register (MDEUISR)	14-58
14.5.4.8	MDEU Interrupt Control Register (MDEUICR)	14-59
14.5.4.9	MDEU ICV Size Register (MDEUICVSR)	14-60
14.5.4.10	MDEU EU-Go Register (MDEUEUG)	14-61
14.5.4.11	MDEU Context Registers	14-61
14.5.4.12	MDEU Key Registers	14-62
14.5.4.13	MDEU FIFO	14-63
14.5.5	Random Number Generator (RNG)	14-63
14.5.5.1	RNG Mode Register (RNGMR)	14-63
14.5.5.2	RNG Data Size Register (RNGDSR)	14-64
14.5.5.3	RNG Reset Control Register (RNGRCR)	14-65
14.5.5.4	RNG Status Register (RNGSR)	14-65
14.5.5.5	RNG Interrupt Status Register (RNGISR)	14-66
14.5.5.6	RNG Interrupt Control Register (RNGICR)	14-67
14.5.5.7	RNG EU-Go Register (RNGEUG)	14-68
14.5.5.8	RNG FIFO	14-68
14.5.6	Advanced Encryption Standard Execution Units (AESU)	14-68
14.5.6.1	AESU Mode Register (AESUMR)	14-68
14.5.6.2	AESU Key Size Register (AESUKSR)	14-71
14.5.6.3	AESU Data Size Register (AESUDSR)	14-71
14.5.6.4	AESU Reset Control Register (AESURCR)	14-72
14.5.6.5	AESU Status Register (AESUSR)	14-73

Contents

Paragraph Number	Title	Page Number
14.5.6.6	AESU Interrupt Status Register (AESUISR)	14-74
14.5.6.7	AESU Interrupt Control Register (AESUICR)	14-75
14.5.6.8	AESU End of Message Register (AESUEMR)	14-76
14.5.6.9	AESU Context Registers	14-77
14.5.6.9.1	Context for CBC Mode	14-78
14.5.6.9.2	Context for Counter Mode	14-78
14.5.6.9.3	Context for SRT Mode	14-78
14.5.6.9.4	Context for CCM Mode	14-79
14.5.6.9.5	AESU Key Registers	14-81
14.5.6.9.6	AESU FIFOs	14-81
14.6	Crypto-Channels	14-81
14.6.1	Crypto-Channel Registers	14-82
14.6.1.1	Crypto-Channel Configuration Register (CCCR)	14-82
14.6.1.2	Crypto-Channel Pointer Status Register (CCPSR)	14-85
14.6.1.3	Crypto-Channel Current Descriptor Pointer Register (CDPR)	14-90
14.6.1.4	Fetch FIFO (FF)	14-90
14.6.1.5	Descriptor Buffer (DB)	14-91
14.6.2	Interrupts	14-92
14.6.2.1	Channel Done Interrupt	14-92
14.6.2.2	Channel Error Interrupt	14-92
14.6.2.3	Channel Reset	14-93
14.7	Controller	14-93
14.7.1	Controller Registers	14-93
14.7.2	EU Assignment Status Register (EUASR)	14-93
14.7.2.1	Interrupt Mask Register (IMR)	14-94
14.7.2.2	Interrupt Status Register (ISR)	14-96
14.7.2.3	Interrupt Clear Register (ICR)	14-96
14.7.2.4	ID Register	14-98
14.7.2.5	Master Control Register (MCR)	14-98
14.7.2.6	EU Access	14-100
14.7.2.7	Multiple EU Assignment	14-100
14.7.2.8	Multiple Channels	14-100
14.7.2.9	Priority Arbitration	14-100
14.7.2.10	Round-Robin Snapshot Arbiters	14-101
14.8	Bus Interface	14-101
14.8.1	Bus Access	14-101
14.8.1.1	Master Read	14-101
14.8.1.2	Master Write	14-102
14.8.1.2.1	Slave Access	14-102
14.8.2	Bus Arbitration Priority	14-102
14.8.3	Snooping by Caches	14-103

Contents

Paragraph Number	Title	Page Number
14.8.4	Interrupts	14-103
14.9	Power–Saving Mode	14-103

Chapter 15
Three-Speed Ethernet Controllers

15.1	Introduction	15-1
15.1.1	Three-Speed Ethernet Controller Overview	15-6
15.2	Features	15-7
15.3	Modes of Operation	15-8
15.4	External Signals Description	15-9
15.4.1	Detailed Signal Descriptions	15-9
15.5	Memory Map/Register Definition	15-12
15.5.1	Top-Level Module Memory Map	15-12
15.5.2	Detailed Memory Map—Control/Status Registers	15-13
15.5.3	Memory-Mapped Register Descriptions	15-19
15.5.3.1	TSEC General Control and Status Registers	15-19
15.5.3.1.1	Interrupt Event Register (IEVENT)	15-19
15.5.3.1.2	Interrupt Mask Register (IMASK)	15-22
15.5.3.1.3	Error Disabled Register (EDIS)	15-24
15.5.3.1.4	Ethernet Control Register (ECNTRL)	15-25
15.5.3.1.5	Minimum Frame Length Register (MINFLR)	15-26
15.5.3.1.6	Pause Time Value Register (PTV)	15-27
15.5.3.1.7	DMA Control Register (DMACTRL)	15-28
15.5.3.1.8	TBI Physical Address Register (TBIPA)	15-29
15.5.3.2	TSEC FIFO Control and Status Registers	15-30
15.5.3.2.1	FIFO Transmit Threshold Register (FIFO_TX_THR)	15-30
15.5.3.2.2	FIFO Transmit Space Available Register (FIFO_TX_SP)	15-31
15.5.3.2.3	FIFO Transmit Starve Register (FIFO_TX_STARVE)	15-31
15.5.3.2.4	FIFO Transmit Starve Shutoff Register (FIFO_TX_STARVE_SHUTOFF)	15-32
15.5.3.3	TSEC Transmit Control and Status Registers	15-33
15.5.3.3.1	Transmit Control Register (TCTRL)	15-33
15.5.3.3.2	Transmit Status Register (TSTAT)	15-34
15.5.3.3.3	TxBD Data Length Register (TBDLEN)	15-34
15.5.3.3.4	Transmit Interrupt Coalescing Configuration Register (TXIC)	15-35
15.5.3.3.5	Current Transmit Buffer Descriptor Pointer Register (CTBPTR)	15-36
15.5.3.3.6	Transmit Buffer Descriptor Pointer Register (TBPTR)	15-36
15.5.3.3.7	Transmit Descriptor Base Address Register (TBASE)	15-37
15.5.3.3.8	Out-of-Sequence TxBD Register (OSTBD)	15-37
15.5.3.3.9	Out-of-Sequence Tx Data Buffer Pointer Register (OSTBDP)	15-39

Contents

Paragraph Number	Title	Page Number
15.5.3.4	TSEC Receive Control and Status Registers	15-40
15.5.3.4.1	Receive Control Register (RCTRL)	15-40
15.5.3.4.2	Receive Status Register (RSTAT)	15-41
15.5.3.4.3	RxBD Data Length Register (RBDLEN)	15-41
15.5.3.4.4	Receive Interrupt Coalescing Configuration Register (RXIC)	15-42
15.5.3.4.5	Current Receive Buffer Descriptor Pointer Register (CRBPTR)	15-43
15.5.3.4.6	Maximum Receive Buffer Length Register (MRBLR)	15-43
15.5.3.4.7	Receive Buffer Descriptor Pointer Register (RBPTR)	15-44
15.5.3.4.8	Receive Descriptor Base Address Register (RBASE)	15-44
15.5.3.5	MAC Functionality	15-45
15.5.3.5.1	Configuring the MAC	15-45
15.5.3.5.2	Controlling CSMA/CD	15-45
15.5.3.5.3	Handling Packet Collisions	15-46
15.5.3.5.4	Controlling Packet Flow	15-46
15.5.3.5.5	Controlling PHY Links	15-47
15.5.3.6	MAC Registers	15-47
15.5.3.6.1	MAC Configuration Register 1 (MACCFG1)	15-48
15.5.3.6.2	MAC Configuration Register 2 (MACCFG2)	15-49
15.5.3.6.3	Inter-Packet Gap/Inter-Frame Gap Register (IPGIFG)	15-51
15.5.3.6.4	Half-Duplex Register (HAFDUP)	15-52
15.5.3.6.5	Maximum Frame Length Register (MAXFRM)	15-53
15.5.3.6.6	MII Management Configuration Register (MIIMCFG)	15-53
15.5.3.6.7	MII Management Command Register (MIIMCOM)	15-54
15.5.3.6.8	MII Management Address Register (MIIMADD)	15-55
15.5.3.6.9	MII Management Control Register (MIIMCON)	15-56
15.5.3.6.10	MII Management Status Register (MIIMSTAT)	15-56
15.5.3.6.11	MII Management Indicator Register (MIIMIND)	15-57
15.5.3.6.12	Interface Status Register (IFSTAT)	15-58
15.5.3.6.13	Station Address Register Part 1 (MACSTNADDR1)	15-58
15.5.3.6.14	Station Address Register Part 2 (MACSTNADDR2)	15-59
15.5.3.7	MIB Registers	15-60
15.5.3.7.1	Transmit and Receive 64-Byte Frame Counter Register (TR64)	15-60
15.5.3.7.2	Transmit and Receive 65- to 127-Byte Frame Counter Register (TR127)	15-61
15.5.3.7.3	Transmit and Receive 128- to 255-Byte Frame Counter Register (TR255)	15-61
15.5.3.7.4	Transmit and Receive 256- to 511-Byte Frame Counter Register (TR511)	15-62
15.5.3.7.5	Transmit and Receive 512- to 1023-Byte Frame Counter Register (TR1K)	15-62

Contents

Paragraph Number	Title	Page Number
15.5.3.7.6	Transmit and Receive 1024- to 1518-Byte Frame Counter Register (TRMAX)	15-63
15.5.3.7.7	Transmit and Receive 1519- to 1522-Byte VLAN Frame Counter Register (TRMGV)	15-63
15.5.3.7.8	Receive Byte Counter Register (RBYT)	15-64
15.5.3.7.9	Receive Packet Counter Register (RPKT)	15-64
15.5.3.7.10	Receive FCS Error Counter Register (RFCS)	15-65
15.5.3.7.11	Receive Multicast Packet Counter Register (RMCA)	15-65
15.5.3.7.12	Receive Broadcast Packet Counter Register (RBCA)	15-66
15.5.3.7.13	Receive Control Frame Packet Counter Register (RXCF)	15-66
15.5.3.7.14	Receive Pause Frame Packet Counter Register (RXPF)	15-67
15.5.3.7.15	Receive Unknown Opcode Packet Counter Register (RXUO)	15-67
15.5.3.7.16	Receive Alignment Error Counter Register (RALN)	15-68
15.5.3.7.17	Receive Frame Length Error Counter Register (RFLR)	15-68
15.5.3.7.18	Receive Code Error Counter Register (RCDE)	15-69
15.5.3.7.19	Receive Carrier Sense Error Counter Register (RCSE)	15-69
15.5.3.7.20	Receive Undersize Packet Counter Register (RUND)	15-70
15.5.3.7.21	Receive Oversize Packet Counter Register (ROVR)	15-70
15.5.3.7.22	Receive Fragments Counter Register (RFRG)	15-71
15.5.3.7.23	Receive Jabber Counter Register (RJBR)	15-71
15.5.3.7.24	Receive Dropped Packet Counter Register (RDRP)	15-72
15.5.3.7.25	Transmit Byte Counter Register (TBYT)	15-72
15.5.3.7.26	Transmit Packet Counter Register (TPKT)	15-73
15.5.3.7.27	Transmit Multicast Packet Counter Register (TMCA)	15-73
15.5.3.7.28	Transmit Broadcast Packet Counter Register (TBCA)	15-74
15.5.3.7.29	Transmit Pause Control Frame Counter Register (TXPF)	15-74
15.5.3.7.30	Transmit Deferral Packet Counter Register (TDFR)	15-75
15.5.3.7.31	Transmit Excessive Deferral Packet Counter Register (TEDF)	15-75
15.5.3.7.32	Transmit Single Collision Packet Counter Register (TSCL)	15-76
15.5.3.7.33	Transmit Multiple Collision Packet Counter Register (TMCL)	15-76
15.5.3.7.34	Transmit Late Collision Packet Counter Register (TLCL)	15-77
15.5.3.7.35	Transmit Excessive Collision Packet Counter Register (TXCL)	15-77
15.5.3.7.36	Transmit Total Collision Counter Register (TNCL)	15-78
15.5.3.7.37	Transmit Drop Frame Counter Register (TDRP)	15-78
15.5.3.7.38	Transmit Jabber Frame Counter Register (TJBR)	15-79
15.5.3.7.39	Transmit FCS Error Counter Register (TFCS)	15-79
15.5.3.7.40	Transmit Control Frame Counter Register (TXCF)	15-80
15.5.3.7.41	Transmit Oversize Frame Counter Register (TOVR)	15-80
15.5.3.7.42	Transmit Undersize Frame Counter Register (TUND)	15-81
15.5.3.7.43	Transmit Fragment Counter Register (TFRG)	15-81
15.5.3.7.44	Carry Register 1 (CAR1)	15-82

Contents

Paragraph Number	Title	Page Number
15.5.3.7.45	Carry Register 2 (CAR2)	15-83
15.5.3.7.46	Carry Mask Register 1 (CAM1)	15-85
15.5.3.7.47	Carry Mask Register 2 (CAM2)	15-86
15.5.3.8	Hash Function Registers	15-87
15.5.3.8.1	Individual Address Registers 0–7 (IADDR*n*)	15-87
15.5.3.8.2	Group Address Registers 0–7 (GADDR*n*)	15-88
15.5.3.9	Attribute Registers	15-89
15.5.3.9.1	Attribute Register (ATTR)	15-89
15.5.4	Ten-Bit Interface (TBI)	15-90
15.5.4.1	TBI MII Set Register Descriptions	15-90
15.5.4.2	Control Register (CR)	15-91
15.5.4.3	Status Register (SR)	15-92
15.5.4.4	AN Advertisement Register (ANA)	15-93
15.5.4.5	AN Link Partner Base Page Ability Register (ANLPBPA)	15-95
15.5.4.6	AN Expansion Register (ANEX)	15-96
15.5.4.7	AN Next Page Transmit Register (ANNPT)	15-97
15.5.4.8	AN Link Partner Ability Next Page Register (ANLPANP)	15-98
15.5.4.9	Extended Status Register (EXST)	15-98
15.5.4.10	Jitter Diagnostics Register (JD)	15-99
15.5.4.11	TBI Control Register (TBICON)	15-100
15.6	Functional Description	15-101
15.6.1	Connecting to Physical Interfaces	15-101
15.6.1.1	Media-Independent Interface (MII)	15-102
15.6.1.2	Gigabit Media-Independent Interface (GMII)	15-102
15.6.1.3	Reduced Gigabit Media-Independent Interface (RGMII)	15-103
15.6.1.4	Ten-Bit Interface (TBI)	15-104
15.6.1.5	Reduced Ten-Bit Interface (RTBI)	15-105
15.6.2	Gigabit Ethernet Channel Operation	15-109
15.6.2.1	Initialization Sequence	15-109
15.6.2.1.1	Hardware Controlled Initialization	15-109
15.6.2.1.2	User Initialization	15-109
15.6.2.2	Soft Reset and Reconfiguring Procedure	15-110
15.6.2.3	Gigabit Ethernet Frame Transmission	15-111
15.6.2.4	Gigabit Ethernet Frame Reception	15-112
15.6.2.5	RMON Support	15-113
15.6.2.6	Frame Recognition	15-113
15.6.2.6.1	Destination Address Recognition	15-114
15.6.2.6.2	Hash Table Algorithm	15-115
15.6.2.7	Flow Control	15-115
15.6.2.8	Interrupt Handling	15-116
15.6.2.8.1	Interrupt Coalescing	15-117

Contents

Paragraph Number	Title	Page Number
15.6.2.8.2	Interrupt Coalescing By Frame Count Threshold	15-117
15.6.2.8.3	Interrupt Coalescing Timer Threshold	15-118
15.6.2.9	Inter-Packet Gap Time	15-119
15.6.2.10	Internal and External Loop Back	15-119
15.6.2.11	Error-Handling Procedure	15-119
15.6.3	Buffer Descriptors	15-120
15.6.3.1	Transmit Data Buffer Descriptor (TxBD)	15-122
15.6.3.2	Receive Buffer Descriptor (RxBD)	15-124
15.7	Initialization/Application Information	15-126
15.7.1	Interface Mode Configuration	15-126
15.7.1.1	MII Interface Mode	15-126
15.7.1.2	GMII Interface Mode	15-130
15.7.1.3	TBI Interface Mode	15-133
15.7.1.4	RGMII Interface Mode	15-137
15.7.1.5	RTBI Interface Mode	15-141

Chapter 16
Universal Serial Bus Interface

16.1	Introduction	16-1
16.1.1	Overview	16-2
16.1.2	Features	16-3
16.1.3	Modes of Operation	16-3
16.2	External Signals	16-4
16.2.1	ULPI Interface	16-7
16.2.2	UTMI Interface	16-8
16.2.3	Serial Interface	16-11
16.2.4	Serial OTG Interface	16-13
16.2.5	PHY Clocks	16-15
16.3	Memory Map/Register Definitions	16-16
16.3.1	Capability Registers	16-18
16.3.1.1	Capability Registers Length (CAPLENGTH)	16-19
16.3.1.2	Host Controller Interface Version (HCIVERSION)	16-19
16.3.1.3	Host Controller Structural Parameters (HCSPARAMS)	16-20
16.3.1.4	Host Controller Capability Parameters (HCCPARAMS)	16-21
16.3.1.5	Device Controller Interface Version (DCIVERSION)—DR Only, Non-EHCI	16-22
16.3.1.6	Device Controller Capability Parameters (DCCPARAMS)—DR Only, Non-EHCI	16-22
16.3.2	Operational Registers	16-23
16.3.2.1	USB Command Register (USBCMD)	16-23

Contents

Paragraph Number	Title	Page Number
16.3.2.2	USB Status Register (USBSTS)	16-26
16.3.2.3	USB Interrupt Enable Register (USBINTR)	16-28
16.3.2.4	Frame Index Register (FRINDEX)	16-30
16.3.2.5	Control Data Structure Segment Register (CTRLDSSEGMENT)	16-31
16.3.2.6	Periodic Frame List Base Address Register (PERIODICLISTBASE)	16-31
16.3.2.7	Device Address Register (DEVICEADDR)—DR-Only, Non-EHCI	16-32
16.3.2.8	Current Asynchronous List Address Register (ASYNCLISTADDR)	16-32
16.3.2.9	Endpoint List Address Register (ENDPOINTLISTADDR)—DR-Only, Non-EHCI	16-33
16.3.2.10	Transaction Translator Asynchronous Buffer Status (ASYNCTTSTS)—MPH-Only, Non-EHCI	16-34
16.3.2.11	Master Interface Data Burst Size Register (BURSTSIZE)—Non-EHCI	16-34
16.3.2.12	Transmit FIFO Tuning Controls Register (TXFILLTUNING)—Non-EHCI	16-35
16.3.2.13	Transaction Translator Transmit FIFO Tuning Controls Register (TXTTFILLTUNING)—MPH-Only, Non-EHCI	16-37
16.3.2.14	ULPI Register Access (ULPI VIEWPORT)	16-37
16.3.2.15	Configure Flag Register (CONFIGFLAG)	16-39
16.3.2.16	Port Status and Control Registers (MPH—PORTSC*n*, DR—PORTSC1)	16-39
16.3.2.17	On-The-Go Status and Control (OTGSC)—DR-Only, Non-EHCI	16-44
16.3.2.18	USB Mode Register (USBMODE)—Non-EHCI	16-47
16.3.2.19	Endpoint Setup Status Register (ENDPTSETUPSTAT)—DR-Only, Non-EHCI	16-48
16.3.2.20	Endpoint Initialization Register (ENDPTPRIME)—DR-Only, Non-EHCI	16-48
16.3.2.21	Endpoint Flush Register (ENDPTFLUSH)—DR-Only, Non-EHCI	16-49
16.3.2.22	Endpoint Status Register (ENDPTSTATUS)—DR-Only, Non-EHCI	16-50
16.3.2.23	Endpoint Complete Register (ENDPTCOMPLETE)—DR-Only, Non-EHCI	16-50
16.3.2.24	Endpoint Control Register 0 (ENDPTCTRL0)—DR-Only, Non-EHCI	16-51
16.3.2.25	Endpoint Control Register *n* (ENDPTCTRL*n*)—DR-Only, Non-EHCI	16-52
16.3.2.26	SNOOP1 and SNOOP2—Non-EHCI	16-53
16.3.2.27	Age Count Threshold Register (AGE_CNT_THRESH)—Non-EHCI	16-54
16.3.2.28	System Interface Control Register (SI_CTRL)—Non-EHCI	16-56
16.3.2.29	Priority Control Register (PRI_CTRL)—Non-EHCI	16-56
16.3.2.30	General Purpose Control Register (CONTROL)—Non-EHCI	16-57
16.4	Functional Description	16-58
16.4.1	System Interface	16-58
16.4.2	DMA Engine	16-58
16.4.3	FIFO RAM Controller	16-58
16.4.4	PHY Interface	16-58
16.5	Host Data Structures	16-59
16.5.1	Periodic Frame List	16-59

Contents

Paragraph Number	Title	Page Number
16.5.2	Asynchronous List Queue Head Pointer	16-61
16.5.3	Isochronous (High-Speed) Transfer Descriptor (iTD)	16-61
16.5.3.1	Next Link Pointer	16-62
16.5.3.2	iTD Transaction Status and Control List	16-63
16.5.3.3	iTD Buffer Page Pointer List (Plus)	16-63
16.5.4	Split Transaction Isochronous Transfer Descriptor (siTD)	16-65
16.5.4.1	Next Link Pointer	16-65
16.5.4.2	siTD Endpoint Capabilities/Characteristics	16-66
16.5.4.3	siTD Transfer State	16-67
16.5.4.4	siTD Buffer Pointer List (Plus)	16-68
16.5.4.5	siTD Back Link Pointer	16-68
16.5.5	Queue Element Transfer Descriptor (qTD)	16-69
16.5.5.1	Next qTD Pointer	16-69
16.5.5.2	Alternate Next qTD Pointer	16-70
16.5.5.3	qTD Token	16-70
16.5.5.4	qTD Buffer Page Pointer List	16-73
16.5.6	Queue Head	16-74
16.5.6.1	Queue Head Horizontal Link Pointer	16-74
16.5.6.2	Endpoint Capabilities/Characteristics	16-75
16.5.6.3	Transfer Overlay	16-77
16.5.7	Periodic Frame Span Traversal Node (FSTN)	16-78
16.5.7.1	FTSN Normal Path Pointer	16-79
16.5.7.2	FSTN Back Path Link Pointer	16-79
16.6	Host Operations	16-79
16.6.1	Host Controller Initialization	16-80
16.6.2	Power Port	16-81
16.6.3	Reporting Over-Current	16-81
16.6.4	Suspend/Resume	16-82
16.6.4.1	Port Suspend/Resume	16-82
16.6.5	Schedule Traversal Rules	16-84
16.6.6	Periodic Schedule Frame Boundaries vs. Bus Frame Boundaries	16-85
16.6.7	Periodic Schedule	16-88
16.6.8	Managing Isochronous Transfers Using iTDs	16-89
16.6.8.1	Host Controller Operational Model for iTDs	16-89
16.6.8.2	Software Operational Model for iTDs	16-91
16.6.8.2.1	Periodic Scheduling Threshold	16-92
16.6.9	Asynchronous Schedule	16-93
16.6.9.1	Adding Queue Heads to Asynchronous Schedule	16-94
16.6.9.2	Removing Queue Heads from Asynchronous Schedule	16-95
16.6.9.3	Empty Asynchronous Schedule Detection	16-97
16.6.9.4	Asynchronous Schedule Traversal: Start Event	16-98

Contents

Paragraph Number	Title	Page Number
16.6.9.5	Reclamation Status Bit (USBSTS Register)	16-98
16.6.10	Managing Control/Bulk/Interrupt Transfers via Queue Heads	16-98
16.6.10.1	Buffer Pointer List Use for Data Streaming with qTDs	16-99
16.6.10.2	Adding Interrupt Queue Heads to the Periodic Schedule	16-101
16.6.10.3	Managing Transfer Complete Interrupts from Queue Heads	16-101
16.6.11	Ping Control	16-102
16.6.12	Split Transactions	16-103
16.6.12.1	Split Transactions for Asynchronous Transfers	16-103
16.6.12.1.1	Asynchronous—Do-Start-Split	16-104
16.6.12.1.2	Asynchronous - Do-Complete-Split	16-104
16.6.12.2	Split Transaction Interrupt	16-105
16.6.12.2.1	Split Transaction Scheduling Mechanisms for Interrupt	16-106
16.6.12.2.2	Host Controller Operational Model for FSTNs	16-108
16.6.12.2.3	Software Operational Model for FSTNs	16-110
16.6.12.2.4	Tracking Split Transaction Progress for Interrupt Transfers	16-111
16.6.12.2.5	Split Transaction Execution State Machine for Interrupt	16-112
16.6.12.2.6	Periodic Interrupt—Do-Start-Split	16-113
16.6.12.2.7	Periodic Interrupt—Do-Complete-Split	16-114
16.6.12.2.8	Managing the QH[FrameTag] Field	16-117
16.6.12.2.9	Rebalancing the Periodic Schedule	16-117
16.6.12.3	Split Transaction Isochronous	16-118
16.6.12.3.1	Split Transaction Scheduling Mechanisms for Isochronous	16-118
16.6.12.3.2	Tracking Split Transaction Progress for Isochronous Transfers	16-122
16.6.12.3.3	Split Transaction Execution State Machine for Isochronous	16-123
16.6.12.3.4	Periodic Isochronous—Do-Start-Split	16-124
16.6.12.3.5	Periodic Isochronous - Do Complete Split	16-126
16.6.12.3.6	Complete-Split for Scheduling Boundary Cases 2a, 2b	16-128
16.6.12.3.7	Split Transaction for Isochronous - Processing Examples	16-129
16.6.13	Port Test Modes	16-131
16.6.14	Interrupts	16-131
16.6.14.1	Transfer/Transaction Based Interrupts	16-132
16.6.14.1.1	Transaction Error	16-132
16.6.14.1.2	Serial Bus Babble	16-133
16.6.14.1.3	Data Buffer Error	16-134
16.6.14.1.4	USB Interrupt (Interrupt on Completion (IOC))	16-134
16.6.14.1.5	Short Packet	16-135
16.6.14.2	Host Controller Event Interrupts	16-135
16.6.14.2.1	Port Change Events	16-135
16.6.14.2.2	Frame List Rollover	16-135
16.6.14.2.3	Interrupt on Async Advance	16-135
16.6.14.2.4	Host System Error	16-135

Contents

Paragraph Number	Title	Page Number
16.7	Device Data Structures—DR Module Only	16-136
16.7.1	Endpoint Queue Head	16-137
16.7.1.1	Endpoint Capabilities/Characteristics	16-138
16.7.1.2	Transfer Overlay	16-139
16.7.1.3	Current dTD Pointer	16-139
16.7.1.4	Set-up Buffer	16-139
16.7.2	Endpoint Transfer Descriptor (dTD)	16-140
16.8	Device Operational Model—DR Module Only	16-142
16.8.1	Device Controller Initialization	16-142
16.8.2	Port State and Control	16-143
16.8.2.1	Bus Reset	16-145
16.8.2.2	Suspend/Resume	16-146
16.8.2.2.1	Suspend Description	16-146
16.8.2.2.2	Suspend Operational Model	16-146
16.8.2.2.3	Resume	16-146
16.8.3	Managing Endpoints	16-147
16.8.3.1	Endpoint Initialization	16-147
16.8.3.1.1	Stalling	16-148
16.8.3.2	Data Toggle	16-149
16.8.3.2.1	Data Toggle Reset	16-149
16.8.3.2.2	Data Toggle Inhibit	16-149
16.8.3.3	Device Operational Model For Packet Transfers	16-149
16.8.3.3.1	Priming Transmit Endpoints	16-149
16.8.3.3.2	Priming Receive Endpoints	16-150
16.8.3.4	Interrupt/Bulk Endpoint Operational Model	16-150
16.8.3.4.1	Interrupt/Bulk Endpoint Bus Response Matrix	16-152
16.8.3.5	Control Endpoint Operation Model	16-152
16.8.3.5.1	Setup Phase	16-152
16.8.3.5.2	Data Phase	16-153
16.8.3.5.3	Status Phase	16-153
16.8.3.5.4	Control Endpoint Bus Response Matrix	16-154
16.8.3.6	Isochronous Endpoint Operational Model	16-154
16.8.3.6.1	Isochronous Pipe Synchronization	16-156
16.8.3.6.2	Isochronous Endpoint Bus Response Matrix	16-156
16.8.4	Managing Queue Heads	16-157
16.8.4.1	Queue Head Initialization	16-158
16.8.4.2	Operational Model For Setup Transfers	16-158
16.8.5	Managing Transfers with Transfer Descriptors	16-159
16.8.5.1	Software Link Pointers	16-159
16.8.5.2	Building a Transfer Descriptor	16-159
16.8.5.3	Executing A Transfer Descriptor	16-160

Contents

Paragraph Number	Title	Page Number
16.8.5.4	Transfer Completion	16-160
16.8.5.5	Flushing/De-priming an Endpoint	16-161
16.8.5.6	Device Error Matrix	16-161
16.8.6	Servicing Interrupts	16-162
16.8.6.1	High-Frequency Interrupts	16-162
16.8.6.2	Low-Frequency Interrupts	16-162
16.8.6.3	Error Interrupts	16-163
16.9	Deviations from the EHCI Specifications	16-163
16.9.1	Embedded Transaction Translator Function—MPH Module Only	16-163
16.9.1.1	Capability Registers	16-164
16.9.1.2	Operational Registers	16-164
16.9.1.3	Discovery	16-164
16.9.1.4	Data Structures	16-165
16.9.1.5	Operational Model	16-165
16.9.1.5.1	Microframe Pipeline	16-165
16.9.1.5.2	Split State Machines	16-166
16.9.1.5.3	Asynchronous Transaction Scheduling and Buffer Management	16-166
16.9.1.5.4	Periodic Transaction Scheduling and Buffer Management	16-167
16.9.1.5.5	Multiple Transaction Translators	16-167
16.9.2	Device Operation—DR Module Only	16-168
16.9.3	Non-Zero Fields the Register File	16-168
16.9.4	SOF Interrupt	16-168
16.9.5	Embedded Design	16-168
16.9.5.1	Frame Adjust Register	16-168
16.9.6	Miscellaneous Variations from EHCI	16-168
16.9.6.1	Programmable Physical Interface Behavior	16-168
16.9.6.2	Discovery	16-169
16.9.6.2.1	Port Reset	16-169
16.9.6.2.2	Port Speed Detection	16-169

Chapter 17
I^2C Interfaces

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

Contents

Paragraph Number	Title	Page Number
17.3.1.1	I2C*n* Address Register (I2C*n*ADR)	17-5
17.3.1.2	I2C*n* Frequency Divider Register (I2C*n*FDR)	17-5
17.3.1.3	I2C*n* Control Register (I2C*n*CR)	17-6
17.3.1.4	I2C*n* Status Register (I2C*n*SR)	17-8
17.3.1.5	I2C*n* Data Register (I2C*n*DR)	17-9
17.3.1.6	Digital Filter Sampling Rate Register (I2C*n*DFSRR)	17-10
17.4	Functional Description	17-10
17.4.1	Transaction Protocol	17-10
17.4.1.1	START Condition	17-11
17.4.1.2	Slave Address Transmission	17-11
17.4.1.3	Repeated START Condition	17-12
17.4.1.4	STOP Condition	17-12
17.4.1.5	Protocol Implementation Details	17-12
17.4.1.5.1	Transaction Monitoring—Implementation Details	17-13
17.4.1.5.2	Control Transfer—Implementation Details	17-13
17.4.1.6	Address Compare—Implementation Details	17-14
17.4.2	Arbitration Procedure	17-14
17.4.2.1	Arbitration Control	17-14
17.4.3	Handshaking	17-15
17.4.4	Clock Control	17-15
17.4.4.1	Clock Synchronization	17-15
17.4.4.2	Input Synchronization and Digital Filter	17-15
17.4.4.2.1	Input Signal Synchronization	17-16
17.4.4.2.2	Filtering of SCL*n* and SDA*n* Lines	17-16
17.4.4.3	Clock Stretching	17-16
17.4.5	Boot Sequencer Mode	17-16
17.4.5.1	Using the Boot Sequencer for Reset Configuration	17-17
17.4.5.2	EEPROM Calling Address	17-17
17.4.5.3	EEPROM Data Format	17-17
17.4.5.4	Boot Sequencer Done Indication	17-20
17.5	Initialization/Application Information	17-20
17.5.1	Interrupt Service Routine Flowchart	17-20
17.5.2	Initialization Sequence	17-22
17.5.3	Generation of START	17-22
17.5.4	Post-Transfer Software Response	17-22
17.5.5	Generation of STOP	17-23
17.5.6	Generation of Repeated START	17-23
17.5.7	Generation of SCL*n* when SDA*n* is Negated	17-23
17.5.8	Slave Mode Interrupt Service Routine	17-23
17.5.8.1	Slave Transmitter and Received Acknowledge	17-24
17.5.8.2	Loss of Arbitration and Forcing of Slave Mode	17-24

Contents

Paragraph Number	Title	Page Number
	Chapter 18 DUART	
18.1	Overview	18-1
18.1.1	Features	18-2
18.1.2	Modes of Operation	18-3
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	18-4
18.3.1	Register Descriptions	18-6
18.3.1.1	Receiver Buffer Registers (URBR1 and URBR2)	18-6
18.3.1.2	Transmitter Holding Registers (UTHR1 and UTHR2)	18-6
18.3.1.3	Divisor Most and Least Significant Byte Registers (UDMB and UDLB)	18-7
18.3.1.4	Interrupt Enable Registers (UIER1 and UIER2)	18-8
18.3.1.5	Interrupt ID Registers (UIIR1 and UIIR2)	18-9
18.3.1.6	FIFO Control Registers (UFCR1 and UFCR2)	18-10
18.3.1.7	Line Control Registers (ULCR1 and ULCR2)	18-11
18.3.1.8	MODEM Control Registers (UMCR1 and UMCR2)	18-13
18.3.1.9	Line Status Registers (ULSR1 and ULSR2)	18-14
18.3.1.10	MODEM Status Registers (UMSR1 and UMSR2)	18-15
18.3.1.11	Scratch Registers (USCR1 and USCR2)	18-16
18.3.1.12	Alternate Function Registers (UAFR1 and UAFR2)	18-16
18.3.1.13	DMA Status Registers (UDSR1 and UDSR2)	18-17
18.4	Functional Description	18-18
18.4.1	Serial Interface	18-19
18.4.1.1	START Bit	18-19
18.4.1.2	Data Transfer	18-19
18.4.1.3	Parity Bit	18-19
18.4.1.4	STOP Bit	18-20
18.4.2	Baud-Rate Generator Logic	18-20
18.4.3	Local Loop-Back Mode	18-20
18.4.4	Errors	18-21
18.4.4.1	Framing Error	18-21
18.4.4.2	Parity Error	18-21
18.4.4.3	Overrun Error	18-21
18.4.5	FIFO Mode	18-21
18.4.5.1	FIFO Interrupts	18-21
18.4.5.2	DMA Mode Select	18-22
18.4.5.3	Interrupt Control Logic	18-22
18.5	DUART Initialization/Application Information	18-22

Contents

Paragraph Number	Title	Page Number
	Chapter 19 Serial Peripheral Interface	
19.1	Overview	19-1
19.2	Introduction	19-2
19.2.1	Features	19-2
19.2.2	SPI Transmission and Reception Process	19-3
19.2.3	Modes of Operation	19-3
19.2.3.1	The SPI as a Master Device	19-3
19.2.3.2	The SPI as a Slave Device	19-4
19.2.3.3	The SPI in Multiple-Master Operation	19-5
19.3	External Signal Descriptions	19-7
19.3.1	Overview	19-7
19.3.2	Detailed Signal Descriptions	19-7
19.4	Memory Map/Register Definition	19-8
19.4.1	Register Descriptions	19-9
19.4.1.1	SPI Mode Register (SPMODE)	19-9
19.4.1.2	SPI Event Register (SPIE)	19-11
19.4.1.3	SPI Mask Register (SPIM)	19-13
19.4.1.4	SPI Command Register (SPCOM)	19-14
19.4.1.5	SPI Transmit Data Hold Register (SPITD)	19-14
19.4.1.6	SPI Receive Data Hold Register (SPIRD)	19-15
19.4.1.6.1	Reverse Mode SPMODE[REV] Examples	19-15
19.5	Initialization/Application Information	19-16
19.5.1	SPI Master Programming Example	19-16
19.5.2	SPI Slave Programming Example	19-16
	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
21.1.1	Overview	21-1
21.1.2	Features	21-2

Contents

Paragraph Number	Title	Page Number
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*n* Direction Register (GP*1*DIR–GP2DIR)	21-3
21.3.2	GPIO*n* Open Drain Register (GP*1*ODR–GP*2*ODR)	21-4
21.3.3	GPIO*n* Data Register (GP*1*DAT–GP*2*DAT)	21-4
21.3.4	GPIO*n* Interrupt Event Register (GP*1*IER–GP2IER)	21-5
21.3.5	GPIO*n* Interrupt Mask Register (GP*1*IMR–GP2IMR)	21-5
21.3.6	GPIO*n* Interrupt Control Register (GP*1*ICR–GP2ICR)	21-6

Chapter 22
Delay Lock Loop (DLL)

22.1	Introduction	22-1
22.2	Overview	22-1
22.2.1	Features	22-2
22.2.2	Modes of Operation	22-2
22.2.3	External Signals	22-2
22.3	Initialization and Application Information	22-2
22.4	Memory Map/Register Definition	22-3
22.4.1	DLL Override Register (DLLOVR)	22-4
22.4.2	DLL Status Register (DLLSR)	22-4
22.4.3	DLL Clock Register (DLLCK)	22-5

Appendix A
Revision History

Figures

Figure Number	Title	Page Number
1-1	MPC8349EA Block Diagram	1-2
1-2	MPC8349EA Integrated e300c1 Core Block Diagram	1-9
1-3	Integrated Security Engine Functional Blocks	1-10
1-4	USB Controllers Port Configuration	1-13
3-1	MPC8349EA Signal Groupings (1 of 2)	3-2
3-2	MPC8349EA Signal Groupings (2 of 2)	3-3
4-1	Power-On Reset Flow	4-7
4-2	Hard Reset Flow	4-8
4-3	Reset Configuration Word Low Register (RCWLR)	4-11
4-4	Reset Configuration Word High Register (RCWHR)	4-14
4-5	Loading Reset Configuration Words from Local Bus	4-23
4-6	Loading Reset Configuration Words from Local Bus (continued)	4-23
4-7	EEPROM Data Format for Reset Configuration Words Preload Command	4-25
4-8	EEPROM Contents	4-26
4-9	Clock Subsystem Block Diagram	4-29
4-10	Reset Status Register (RSR)	4-33
4-11	Reset Mode Register (RMR)	4-34
4-12	Reset Protection Register (RPR)	4-35
4-13	Reset Control Register (RCR)	4-36
4-14	Reset Control Enable Register (RCER)	4-36
4-15	System PLL Mode Register	4-37
4-16	Output Clock Control Register (OCCR)	4-38
4-17	System Clock Control Register (SCCR)	4-40
4-18	MCK Clock Register (MCKENR)	4-41
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 *n* Base Address Registers (LBLAWBAR0–LBLAWBAR3)	5-7
5-5	LBC Local Access Window *n* Attributes Registers (LBLAWAR0–LBLAWAR3)	5-8
5-6	PCI Local Access Window *n* Base Address Registers (PCILAWBAR0–PCILAWBAR1)	5-9
5-7	PCI Local Access Window *n* Attributes Registers (PCILAWAR0–PCILAWAR1)	5-10
5-8	DDR Local Access Window *n* Base Address Registers (DDRLAWBAR0–DDRLAWBAR1)	5-12
5-9	DDR Local Access Window *n* Attributes Registers (DDRLAWAR0–DDRLAWAR1)	5-13
5-10	System General Purpose Register Low (SGPRL)	5-17
5-11	System General Purpose Register High (SGPRH)	5-17
5-12	System Part and Revision ID Register (SPRIDR)	5-18
5-13	System Priority Configuration Register (SPCR)	5-19

Figures

Figure Number	Title	Page Number
5-14	System I/O Configuration Register Low (SICRL)	5-21
5-15	System I/O Configuration Register High (SICRH)	5-25
5-16	DDR Control Driver Register (DDRCDR)	5-29
5-17	DDR Debug Status Register (DDRDSR)	5-30
5-18	Software Watchdog Timer High-Level Block Diagram	5-32
5-19	System Watchdog Control Register (SWCRR)	5-33
5-20	System Watchdog Count Register (SWCNR)	5-34
5-21	System Watchdog Service Register (SWSRR)	5-35
5-22	Software Watchdog Timer Service State Diagram	5-36
5-23	Software Watchdog Timer Functional Block Diagram	5-36
5-24	Real Time Clock Module High Level Block Diagram	5-38
5-25	Real Time Counter Control Register (RTCNR)	5-40
5-26	Real Time Counter Load Register (RTLDR)	5-41
5-27	Real Time Counter Prescale Register (RTPSR)	5-41
5-28	Real Time Counter Register (RTCTR)	5-42
5-29	Real Time Counter Event Register (RTEVR)	5-42
5-30	Real Time Counter Alarm Register (RTALR)	5-43
5-31	Real Time Clock Module Functional Block Diagram	5-44
5-32	Periodic Interval Timer High Level Block Diagram	5-46
5-33	Periodic Interval Timer Control Register (PTCNR)	5-47
5-34	Periodic Interval Timer Load Register (PTLDR)	5-48
5-35	Periodic Interval Timer Prescale Register (PTPSR)	5-49
5-36	Periodic Interval Timer Counter Register (PTCTR)	5-49
5-37	Periodic Interval Timer Event Register (PTEVR)	5-50
5-38	Periodic Interval Timer Functional Block Diagram	5-51
5-39	Global Timers Block Diagram	5-52
5-40	Global Timers Configuration Register 1 (GTCFR1)	5-57
5-41	Global Timers Configuration Register 2 (GTCFR2)	5-59
5-42	Global Timers Mode Registers (GTMDR1–GTMDR4)	5-60
5-43	Global Timers Reference Registers (GTRFR1–GTRFR4)	5-62
5-44	Global Timers Capture Registers (GTCPR1–GTCPR4)	5-62
5-45	Global Timers Counter Registers (GTCNR1—GTCNR4)	5-63
5-46	Global Timers Event Registers (GTEVR1—GTEVR4)	5-63
5-47	Global Timers Prescale Registers (GTPSR1–GTPSR4)	5-64
5-48	Timers Non-Cascaded Mode Block Diagram	5-66
5-49	Timer Pair-Cascaded Mode Block Diagram	5-67
5-50	Timers Super-Cascaded Mode Block Diagram	5-67
5-51	Power Management Controller Configuration Register	5-69
5-52	Power Management Controller Event Register	5-70
5-53	Power Management Controller Mask Register	5-71
6-1	Arbiter Configuration Register (ACR)	6-2

Figures

Figure Number	Title	Page Number
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
6-5	Arbiter Mask Register (AMR)	6-7
6-6	Arbiter Event Attributes Register (AEATR)	6-8
6-7	Arbiter Event Address Register (AEADR)	6-9
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	6-12
7-1	e300c1 Core Block Diagram	7-2
7-2	e300 Programming Model—Registers	7-14
7-3	e300c1 Data Cache Organization	7-28
7-4	Core Interface	7-36
8-1	Interrupt Sources Block Diagram	8-3
8-2	System Global Interrupt Configuration Register (SICFR)	8-8
8-3	System Global Interrupt Vector Register (SIVCR)	8-9
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 D Priority Register (SIPRR_D)	8-14
8-8	System Internal Interrupt Mask Register (SIMSR_H)	8-15
8-9	System Internal Interrupt Mask Register (SIMSR_L)	8-16
8-10	System Internal Interrupt Control Register (SICNR)	8-17
8-11	System External Interrupt Pending Register (SEPNR)	8-18
8-12	System Mixed Interrupt Group A Priority Register (SMPRR_A)	8-18
8-13	System Mixed Interrupt Group B Priority Register (SMPRR_B)	8-19
8-14	System External Interrupt Mask Register (SEMSR)	8-20
8-15	System External Interrupt Control Register (SECNR)	8-21
8-16	System Error Status Register (SERSR)	8-22
8-17	System Error Mask Register (SERMR)	8-24
8-18	System Error Control Register (SERCR)	8-24
8-19	System Internal Interrupt Force Register (SIFCR_H)	8-25
8-20	System Internal Interrupt Force Register (SIFCR_L)	8-25
8-21	System External Interrupt Force Register (SEFCR)	8-26
8-22	System Error Status Register (SERFR)	8-26
8-23	System Critical Interrupt Vector Register (SCVCR)	8-27
8-24	System Management Interrupt Vector Register (SMVCR)	8-28
8-25	Interrupt Structure	8-29
8-26	DDR Interrupt Request Masking	8-35
9-1	DDR Memory Controller Simplified Block Diagram	9-2
9-2	Chip Select Bounds Registers (CS*n*_BNDS)	9-11

Figures

Figure Number	Title	Page Number
9-3	Chip Select Configuration Register (CS*n*_CONFIG)	9-11
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)	9-20
9-9	DDR SDRAM Control Configuration Register 2 (DDR_SDRAM_CFG_2)	9-22
9-10	DDR SDRAM Mode Configuration Register (DDR_SDRAM_MODE)	9-24
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)	9-30
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)	9-31
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)	9-32
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)	9-33
9-24	Memory Data Path Read Capture ECC Register (CAPTURE_ECC)	9-33
9-25	Memory Error Detect Register (ERR_DETECT)	9-34
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	DDR Memory Controller Block Diagram	9-39
9-32	Typical Dual Data Rate SDRAM Internal Organization	9-40
9-33	Typical DDR SDRAM Interface Signals	9-41
9-34	Example 256-Mbyte DDR SDRAM Configuration With ECC	9-42
9-35	DDR SDRAM Burst Read Timing—ACTTORW = 3, $\overline{\text{MCAS}}$ Latency = 2	9-53
9-36	DDR SDRAM Single-Beat (Double Word) Write Timing—ACTTORW = 3	9-53
9-37	DDR SDRAM 4-Beat Burst Write Timing—ACTTORW = 4	9-54
9-38	DDR SDRAM Clock Distribution Example for x8 DDR SDRAMs	9-55
9-39	DDR SDRAM Mode-Set Command Timing	9-55
9-40	Registered DDR1 SDRAM DIMM Burst Write Timing	9-56
9-41	Write Timing Adjustments Example for Write Latency = 1	9-57
9-42	DDR SDRAM Bank Staggered Auto Refresh Timing	9-58
9-43	DDR SDRAM Power-Down Mode	9-59

Figures

Figure Number	Title	Page Number
9-44	DDR SDRAM Self-Refresh Entry Timing	9-60
9-45	DDR SDRAM Self-Refresh Exit Timing	9-60
10-1	Local Bus Controller Block Diagram	10-1
10-2	Base Registers (BR*n*)	10-11
10-3	Option Registers (OR*n*) in GPCM Mode	10-13
10-4	Option Registers (OR*n*) in UPM Mode	10-16
10-5	Option Registers (OR*n*) in SDRAM Mode	10-17
10-6	UPM Memory Address Register (MAR)	10-18
10-7	UPM Mode Registers (M*n*MR)	10-19
10-8	Memory Refresh Timer Prescaler Register (MRTPR)	10-21
10-9	UPM Data Register (MDR)	10-22
10-10	Local Bus SDRAM Machine Mode Register (LSDMR)	10-22
10-11	UPM Refresh Timer (LURT)	10-24
10-12	LSRT SDRAM Refresh Timer (LSRT)	10-25
10-13	Transfer Error Status Register (LTESR)	10-26
10-14	Transfer Error Check Disable Register (LTEDR)	10-27
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	Local Bus Configuration Register	10-29
10-19	Clock Ratio Register (LCRR)	10-30
10-20	Basic Operation of Memory Controllers in the LBC	10-32
10-21	Example of 8-Bit GPCM Writing 32 Bytes to Address 0x5420	10-34
10-22	Basic LBC Bus Cycle with LALE, TA, and $\overline{\text{LCS}}$*n*	10-34
10-23	Local Bus to GPCM Device Interface	10-36
10-24	GPCM Basic Read Timing (XACS = 0, ACS = 1x, TRLX = 0, CLKDIV = 4,8)	10-36
10-25	GPCM Basic Write Timing (XACS = 0, ACS = 00, CSNT = 1, SCY = 1, TRLX = 0, CLKDIV = 4, 8)	10-41
10-26	GPCM Relaxed Timing Read (XACS = 0, ACS = 1x, SCY = 1, CSNT = 0, TRLX = 1, CLKDIV = 4, 8)	10-42
10-27	GPCM Relaxed Timing Back-to-Back Writes (XACS = 0, ACS = 1x, SCY = 0, CSNT = 0, TRLX = 1, CLKDIV = 4, 8)	10-43
10-28	GPCM Relaxed Timing Write (XACS = 0, ACS = 10, SCY = 0, CSNT = 1, TRLX = 1, CLKDIV = 4, 8)	10-44
10-29	GPCM Relaxed Timing Write (XACS = 0, ACS = 00, SCY = 1, CSNT = 1, TRLX = 1, CLKDIV = 4, 8)	10-44
10-30	GPCM Read Followed by Read (TRLX = 0, EHTR = 0, Fastest Timing)	10-45
10-31	GPCM Read Followed by Write (TRLX = 0, EHTR = 1, One-Cycle Extended Hold Time on Reads)	10-46
10-32	External Termination of GPCM Access	10-47

Figures

Figure Number	Title	Page Number
10-33	Connection to a 32-Bit SDRAM with 12 Address Lines	10-49
10-34	SDRAM Address Multiplexing	10-52
10-35	PRETOACT = 2 (2 Clock Cycles)	10-53
10-36	ACTTORW = 2 (2 Clock Cycles)	10-53
10-37	CL = 2 (2 Clock Cycles)	10-54
10-38	WRC = 2 (2 Clock Cycles)	10-54
10-39	RFRC = 4 (6 Clock Cycles)	10-54
10-40	BUFCMD = 1, LCRR[BUFCMDC] = 2	10-55
10-41	SDRAM Single-Beat Read, Page Closed, CL = 3	10-55
10-42	SDRAM Single-Beat Read, Page Hit, CL = 3	10-55
10-43	SDRAM Two-Beat Burst Read, Page Closed, CL = 3	10-56
10-44	SDRAM Four-Beat Burst Read, Page Miss, CL = 3	10-56
10-45	SDRAM Single-Beat Write, Page Hit	10-56
10-46	SDRAM Three-Beat Write, Page Closed	10-56
10-47	SDRAM Read-after-Read Pipelined, Page Hit, CL = 3	10-57
10-48	SDRAM Write-after-Write Pipelined, Page Hit	10-57
10-49	SDRAM Read-after-Write Pipelined, Page Hit	10-57
10-50	SDRAM MODE-SET Command	10-58
10-51	SDRAM Bank-Staggered Auto Refresh Timing	10-59
10-52	User-Programmable Machine Functional Block Diagram	10-59
10-53	RAM Array Indexing	10-60
10-54	Memory Refresh Timer Request Block Diagram	10-61
10-55	UPM Clock Scheme for LCRR[CLKDIV] = 2	10-65
10-56	UPM Clock Scheme for LCRR[CLKDIV] = 4 or 8	10-65
10-57	UPM RAM Array and Signal Generation	10-65
10-58	UPM RAM Word Field Descriptions	10-66
10-59	LCSn Signal Selection	10-69
10-60	LBS Signal Selection	10-70
10-61	UPM Read Access Data Sampling	10-72
10-62	Effect of LUPWAIT Signal	10-73
10-63	Single-Beat Read Access to FPM DRAM	10-75
10-64	Single-Beat Write Access to FPM DRAM	10-76
10-65	Burst Read Access to FPM DRAM Using LOOP (Two Beats Shown)	10-77
10-66	Refresh Cycle (CBR) to FPM DRAM	10-78
10-67	Exception Cycle	10-79
10-68	Multiplexed Address/Data Bus	10-80
10-69	Local Bus Peripheral Hierarchy	10-81
10-70	Local Bus Peripheral Hierarchy for Very High Bus Speeds	10-82
10-71	GPCM Address Timings	10-82
10-72	GPCM Data Timings	10-83
10-73	Interface to Different Port-Size Devices	10-85

Figures

Figure Number	Title	Page Number
10-74	128-Mbyte SDRAM Diagram	10-89
10-75	Parity Support for SDRAM	10-93
10-76	Interface to ZBT SRAM	10-94
10-77	Interface to MSC8122 DSI in Asynchronous Mode	10-96
10-78	Asynchronous Write to MSC8122 DSI	10-97
10-79	Asynchronous Read from MSC8122 DSI	10-98
10-80	Interface to MSC8122 DSI in Synchronous Mode	10-99
10-81	UPM Synchronization Cycle	10-100
10-82	Synchronous Single Write to MSC8122 DSI	10-101
10-83	Synchronous Single Read from MSC8122 DSI	10-102
10-84	Synchronous Burst Write to MSC8122 DSI	10-103
10-85	Synchronous Burst Read from MSC8122 DSI	10-104
11-1	I/O Sequencer Block Diagram	11-1
11-2	PCI Outbound Translation Address Registers (POTARn)	11-3
11-3	PCI Outbound Base Address Registers (POBARn)	11-3
11-4	PCI Outbound Comparison Mask Registers (POCMRn)	11-4
11-5	Power Management Control Register (PMCR)	11-5
11-6	Discard Timer Control Register (DTCR)	11-6
11-7	Outbound PCI Memory Address Translation	11-8
12-1	DMA/Messaging Unit Block Diagram	12-1
12-2	Outbound Message Interrupt Status Register (OMISR)	12-5
12-3	Outbound Message Interrupt Mask Register (OMIMR)	12-6
12-4	Inbound Message Registers (IMR0, IMR1)	12-7
12-5	Outbound Message Registers (OMR0–OMR1)	12-7
12-6	Outbound Doorbell Register (ODR)	12-8
12-7	Inbound Doorbell Register (IDR)	12-9
12-8	Inbound Message Interrupt Status Register (IMISR)	12-10
12-9	Inbound Message Interrupt Mask Register (IMIMR)	12-11
12-10	DMA Mode Registers (DMAMRn)	12-12
12-11	DMA Status Register (DMASRn)	12-14
12-12	DMA Current Descriptor Address Register (DMACDAR*n*)	12-15
12-13	DMA Source Address Register (DMASAR*n*)	12-16
12-14	DMA Destination Address Register (DMADAR*n*)	12-16
12-15	DMA Byte Count Register (DMABCR*n*)	12-17
12-16	DMA Next Descriptor Address Register (DMANDAR*n*)	12-17
12-17	DMA General Status Register (DMAGSR)	12-18
12-18	DMA Controller Block Diagram	12-20
12-19	DMA Chain of Segment Descriptors	12-23
13-1	PCI Controller Block Diagram	13-2
13-2	PCI Interface External Signals	13-6
13-3	PCI_CONFIG_ADDRESS Register	13-16

Figures

Figure Number	Title	Page Number
13-4	PCI_CONFIG_DATA	13-18
13-5	PCI Error Status Register (PCI_ESR)	13-19
13-6	PCI Error Capture Disable Register (PCI_ECDR)	13-20
13-7	PCI Error Enable Register (PCI_EER)	13-20
13-8	PCI Error Attributes Capture Register (PCI_EATCR)	13-21
13-9	PCI Error Address Capture Register (PCI_EACR)	13-23
13-10	PCI Error Extended Address Capture Register (PCI_EEACR)	13-23
13-11	PCI Error Data Low Capture Register (PCI_EDLCR)	13-24
13-12	PCI Error Data High Capture Register (PCI_EDHCR)	13-24
13-13	PCI General Control Register (PCI_GCR)	13-25
13-14	PCI Error Control Register (PCI_ECR)	13-25
13-15	PCI General Status Register (PCI_GSR)	13-26
13-16	PCI Inbound Translation Address Registers (PITARn)	13-27
13-17	PCI Inbound Base Address Registers (PIBARn)	13-27
13-18	PCI Inbound Extended Base Address Registers (PIEBARn)	13-28
13-19	PCI Inbound Window Attribute Registers (PIWARn)	13-28
13-20	Vendor ID Configuration Register	13-31
13-21	Device ID Configuration Register	13-31
13-22	PCI Command Configuration Register	13-32
13-23	PCI Status Configuration Register	13-33
13-24	Revision ID Configuration Register	13-34
13-25	Standard Programming Interface Configuration Register	13-34
13-26	Subclass Code Configuration Register	13-35
13-27	Base Class Code Configuration Register	13-35
13-28	Cache Line Size Configuration Register	13-36
13-29	Latency Timer Configuration Register	13-36
13-30	Header Type Configuration Register	13-37
13-31	BIST Control Configuration Register	13-37
13-32	PIMMR Base Address Configuration Register	13-37
13-33	GPL Base Address Register 0	13-38
13-34	GPL Base Address Registers 1–2	13-38
13-35	GPL Extended Base Address Registers 1–2	13-39
13-36	Subsystem Vendor ID Configuration Register	13-40
13-37	Subsystem Device ID Configuration Register	13-40
13-38	Capabilities Pointer Configuration Register	13-40
13-39	Interrupt Line Configuration Register	13-41
13-40	Interrupt Pin Register	13-41
13-41	Minimum Grant Configuration Register	13-41
13-42	Maximum Latency Configuration Register	13-42
13-43	PCI Function Configuration Register	13-42
13-44	PCI Arbiter Control Register (PCIACR)	13-43

Figures

Figure Number	Title	Page Number
13-45	Hot Swap Register Block	13-44
13-46	PCI Arbitration Example	13-46
13-47	Single Beat Read Example	13-51
13-48	Burst Read Example	13-51
13-49	Single Beat Write Example	13-52
13-50	Burst Write Example	13-52
13-51	Target-Initiated Terminations	13-53
13-52	PCI Parity Operation	13-59
13-53	Inbound PCI Memory Address Translation	13-60
14-1	SEC Connected to MPC8349EA Internal Bus	14-3
14-2	SEC Functional Modules	14-3
14-4	Header Dword	14-16
14-3	Descriptor Format	14-16
14-5	Pointer Dword	14-20
14-6	Link Table Entry Format	14-21
14-7	Descriptors, Link Tables, and Data Parcels	14-23
14-8	PKEU Mode Register	14-26
14-9	PKEU Key Size Register	14-28
14-10	PKEU AB Size Register	14-29
14-11	PKEU Data Size Register (PKEUDSR)	14-29
14-12	PKEU Reset Control Register (PKEURCR)	14-29
14-13	PKEU Status Register (PKEUSR)	14-30
14-14	PKEU Interrupt Status Register (PKEUISR)	14-31
14-15	PKEU Interrupt Control Register (PKEUICR)	14-32
14-16	PKEU EU-Go Register (PKEUEUG)	14-34
14-17	DEU Mode Register (DEUMR)	14-35
14-18	DEU Key Size Register (DEUKSR)	14-36
14-19	DEU Data Size Register (DEUDSR)	14-37
14-20	DEU Reset Control Register (DEURCR)	14-37
14-21	DEU Status Register (DEUSR)	14-38
14-22	DEU Interrupt Status Register (DEUISR)	14-38
14-23	DEU Interrupt Control Register (DEUICR)	14-40
14-24	DEU EU-Go Register (DEUEUG)	14-42
14-25	AFEU Mode Register (AFEUMR)	14-43
14-26	AFEU Key Size Register (AFEUKSR)	14-44
14-27	AFEU Data Size Register	14-45
14-28	AFEU Reset Control Register (AFEURCR)	14-46
14-29	AFEU Status Register	14-46
14-30	AFEU Interrupt Status Register (AFEUISR)	14-47
14-31	AFEU Interrupt Control Register (AFEUICR)	14-49
14-32	AFEU End of Message Register (AFEUEMR)	14-50

Figures

Figure Number	Title	Page Number
14-33	MDEU Mode Register (MDEUMR) in ‘Old’ Configuration	14-52
14-34	MDEU Mode Register (MDEUMR) in ‘New’ Configuration	14-53
14-35	MDEU Key Size Register (MDEUKSR)	14-55
14-36	MDEU Data Size Register (MDEUDSR)	14-56
14-37	MDEU Reset Control Register (MDEURCR)	14-56
14-38	MDEU Status Register	14-57
14-39	MDEU Interrupt Status Register (MDEUISR)	14-58
14-40	MDEU Interrupt Control Register (MDEUICR)	14-59
14-41	MDEU ICV Size Register	14-61
14-42	MDEU EU-Go Register	14-61
14-43	MDEU Context Register	14-62
14-44	RNG Mode Register	14-64
14-45	RNG Data Size Register	14-64
14-46	RNG Reset Control Register	14-65
14-47	RNG Status Register	14-65
14-48	RNG Interrupt Status Register	14-66
14-49	RNG Interrupt Control Register	14-67
14-50	RNG EU-Go Register	14-68
14-51	AESU Mode Register	14-69
14-52	AESU Key Size Register	14-71
14-53	AESU Data Size Register	14-72
14-54	AESU Reset Control Register	14-72
14-55	AESU Status Register	14-73
14-56	AESU Interrupt Status Register	14-74
14-57	AESU Interrupt Control Register	14-75
14-58	AESU End of Message Register	14-77
14-59	AESU Context Register	14-77
14-60	Crypto-Channel Configuration Register (CCCR)	14-82
14-61	Header Dword Writeback Format	14-84
14-62	Crypto-Channel Pointer Status Register	14-85
14-63	Crypto-Channel Current Descriptor Pointer Register	14-90
14-64	Fetch FIFO	14-91
14-65	Data Packet Descriptor Buffer	14-92
14-66	EU Assignment Status Register (EUASR)	14-94
14-67	Interrupt Mask Register	14-95
14-68	Interrupt Status Register	14-96
14-69	Interrupt Clear Register	14-97
14-70	ID Register	14-98
14-71	Master Control Register	14-98
15-1	Ethernet Protocol in Relation to the OSI Protocol Stack	15-2
15-2	802.3z and 802.3ab Physical Standards	15-3

Figures

Figure Number	Title	Page Number
15-3	Ethernet/IEEE 802.3 Frame Structure	15-3
15-4	Ethernet/IEEE 802.3 Frame Structure with More Details	15-4
15-5	TSEC Block Diagram	15-6
15-6	IEVENT Register Definition	15-19
15-7	IMASK Register Definition	15-22
15-8	Error Disabled Register (EDIS)	15-24
15-9	ECNTRL Register Definition	15-25
15-10	MINFLR Register Definition	15-26
15-11	PTV Register Definition	15-27
15-12	DMACTRL Register Definition	15-28
15-13	TBIPA Register Definition	15-29
15-14	FIFO_TX_THR Register Definition	15-31
15-15	FIFO_TX_SP Register Definition	15-31
15-16	FIFO_TX_STARVE Register Definition	15-32
15-17	FIFO_TX_STARVE_SHUTOFF Register Definition	15-32
15-18	TCTRL Register Definition	15-33
15-19	TSTAT Register Definition	15-34
15-20	TBDLEN Register Definition	15-34
15-21	TXIC Register Definition	15-35
15-22	CTBPTR Register Definition	15-36
15-23	TBPTR Register Definition	15-36
15-24	TBASE Register Definition	15-37
15-25	OSTBD Register Definition	15-38
15-26	OSTDBP Register Definition	15-39
15-27	RCTRL Register Definition	15-40
15-28	RSTAT Register Definition	15-41
15-29	RBDLEN Register Definition	15-41
15-30	RXIC Register Definition	15-42
15-31	CRBPTR Register Definition	15-43
15-32	MRBL Register Definition	15-43
15-33	RBPTR Register Definition	15-44
15-34	RBASE Register Definition	15-45
15-35	MACCFG1 Register Definition	15-48
15-36	MACCFG2 Register Definition	15-49
15-37	IPGIFG Register Definition	15-51
15-38	Half-Duplex Register Definition	15-52
15-39	Maximum Frame Length Register Definition	15-53
15-40	MII Management Configuration Register Definition	15-53
15-41	MIIMCOM Register Definition	15-54
15-42	MIIMADD Register Definition	15-55
15-43	MII Management Control Register Definition	15-56

Figures

Figure Number	Title	Page Number
15-44	MIIMSTAT Register Definition	15-56
15-45	MII Management Indicator Register Definition	15-57
15-46	IFSTAT Register Definition	15-58
15-47	MACSTNADDR1 Register Definition	15-59
15-48	MACSTNADDR2 Register Definition	15-59
15-49	Transmit and Receive 64-Byte Frame Register Definition	15-60
15-50	Transmit and Receive 65- to 127-Byte Frame Register Definition	15-61
15-51	Transmit and Received 128- to 255-Byte Frame Register Definition	15-61
15-52	Transmit and Received 256- to 511-Byte Frame Register Definition	15-62
15-53	Transmit and Received 512- to 1023-Byte Frame Register Definition	15-62
15-54	Transmit and Received 1024- to 1518-Byte Frame Register Definition	15-63
15-56	Transmit and Received 1519- to 1522-Byte VLAN Frame Register Definition	15-63
15-57	Receive Byte Counter Register Definition	15-64
15-58	Receive Packet Counter Register Definition	15-64
15-59	Receive FCS Error Counter Register Definition	15-65
15-60	Receive Multicast Packet Counter Register Definition	15-65
15-61	Receive Broadcast Packet Counter Register Definition	15-66
15-62	Receive Control Frame Packet Counter Register Definition	15-66
15-63	Receive Pause Frame Packet Counter Register Definition	15-67
15-64	Receive Unknown Opcode Packet Counter Register Definition	15-67
15-65	Receive Alignment Error Counter Register Definition	15-68
15-66	Receive Frame Length Error Counter Register Definition	15-68
15-67	Receive Code Error Counter Register Definition	15-69
15-68	Receive Carrier Sense Error Counter Register Definition	15-69
15-69	Receive Undersize Packet Counter Register Definition	15-70
15-70	Receive Oversize Packet Counter Register Definition	15-70
15-71	Receive Fragments Counter Register Definition	15-71
15-72	Receive Jabber Counter Register Definition	15-71
15-73	Receive Dropped Packet Counter Register Definition	15-72
15-74	Transmit Byte Counter Register Definition	15-72
15-75	Transmit Packet Counter Register Definition	15-73
15-76	Transmit Multicast Packet Counter Register Definition	15-73
15-77	Transmit Broadcast Packet Counter Register Definition	15-74
15-78	Transmit Pause Control Frame Counter Register Definition	15-74
15-79	Transmit Deferral Packet Counter Register Definition	15-75
15-80	Transmit Excessive Deferral Packet Counter Register Definition	15-75
15-81	Transmit Single Collision Packet Counter Register Definition	15-76
15-82	Transmit Multiple Collision Packet Counter Register Definition	15-76
15-83	Transmit Late Collision Packet Counter Register Definition	15-77
15-84	Transmit Excessive Collision Packet Counter Register Definition	15-77
15-85	Transmit Total Collision Counter Register Definition	15-78

Figures

Figure Number	Title	Page Number
15-86	Transmit Drop Frame Counter Register Definition	15-78
15-87	Transmit Jabber Frame Counter Register Definition	15-79
15-88	Transmit FCS Error Counter Register Definition	15-79
15-89	Transmit Control Frame Counter Register Definition	15-80
15-90	Transmit Oversized Frame Counter Register Definition	15-80
15-91	Transmit Undersize Frame Counter Register Definition	15-81
15-92	Transmit Fragment Counter Register Definition	15-81
15-93	Carry Register 1 (CAR1) Register Definition	15-82
15-94	Carry Register 2 (CAR2) Register Definition	15-83
15-95	Carry Mask Register 1 (CAM1) Register Definition	15-85
15-96	Carry Mask Register 2 (CAM2) Register Definition	15-86
15-97	IADDR*n* Register Definition	15-88
15-98	GADDR*n* Register Definition	15-88
15-99	ATTR Register Definition	15-89
15-100	Control Register Definition	15-91
15-101	Status Register Definition	15-92
15-102	AN Advertisement Register Definition	15-93
15-103	AN Link Partner Base Page Ability Register Definition	15-95
15-104	AN Expansion Register Definition	15-96
15-105	AN Next Page Transmit Register Definition	15-97
15-106	AN Link Partner Ability Next Page Register Definition	15-98
15-107	Extended Status Register Definition	15-98
15-108	Jitter Diagnostics Register Definition	15-99
15-109	TBI Control Register Definition	15-100
15-110	TSEC-MII Connection	15-102
15-111	TSEC-GMII Connection	15-103
15-112	TSEC-RGMII Connection	15-104
15-113	TSEC-TBI Connection	15-105
15-114	TSEC-RTBI Connection	15-106
15-115	Ethernet Address Recognition Flowchart	15-114
15-116	Example of TSEC Memory Structure for BD	15-121
15-117	Buffer Descriptor Ring	15-121
15-118	Transmit Buffer Descriptor	15-122
15-119	Receive Buffer Descriptor	15-124
16-1	USB Interface Block Diagram	16-2
16-2	Capability Registers Length (CAPLENGTH)	16-19
16-3	Host Controller Interface Version (HCIVERSION)	16-19
16-4	Host Controller Structural Parameters (HCSPARAMS)	16-20
16-5	Host Control Capability Parameters (HCCPARAMS)	16-21
16-6	Device Interface Version (DCIVERSION)—DR-Only	16-22
16-7	Device Control Capability Parameters (DCCPARAMS)—DR-Only	16-22

Figures

Figure Number	Title	Page Number
16-8	USB Command Register (USBCMD)	16-23
16-9	USB Status Register (USBSTS)	16-26
16-10	USB Interrupt Enable (USBINTR)	16-28
16-11	USB Frame Index (FRINDEX)	16-30
16-12	Periodic Frame List Base Address (PERIODICLISTBASE)	16-32
16-13	Device Address (DEVICEADDR)—DR-Only	16-32
16-14	Current Asynchronous List Address (ASYNCLISTADDR)	16-33
16-15	Endpoint List Address (ENDPOINTLISTADDR)—DR-Only	16-33
16-16	TT Asynchronous Buffer Status (ASYNCTTSTS)—MPH-Only	16-34
16-17	Master Interface Data Burst Size (BURSTSIZE)	16-34
16-18	Transmit FIFO Tuning Controls (TXFILLTUNING)	16-36
16-19	TT Transmit FIFO Tuning Controls (TXTTFILLTUNING)—MPH-Only	16-37
16-20	ULPI Register Access (ULPI VIEWPORT)	16-38
16-21	Configure Flag Register (CONFIGFLAG)	16-39
16-22	Port Status and Control (PORTSC*n*)	16-40
16-23	OTG Status Control (OTGSC)—DR-Only	16-45
16-24	USB Mode (USBMODE)	16-47
16-25	Endpoint Setup Status (ENDPTSETUPSTAT)—DR-Only	16-48
16-26	Endpoint Initialization (ENDPTPRIME)—DR-Only	16-48
16-27	Endpoint Flush (ENDPTFLUSH)—DR-Only	16-49
16-28	Endpoint Status (ENDPTSTATUS)—DR-Only	16-50
16-29	Endpoint Complete (ENDPTCOMPLETE)—DR-Only	16-50
16-30	Endpoint Control 0 (ENDPTCTRL0)—DR-Only	16-51
16-31	Endpoint Control 1 to 5 (ENDPTCTRL*n*)—DR-Only	16-52
16-32	Snoop 1 and Snoop 2 (SNOOP*n*)	16-54
16-33	Age Count Threshold (AGE_CNT_THRESH)	16-55
16-34	System Interface Control Register (SI_CTRL)	16-56
16-35	Priority Control (PRI_CTRL)	16-56
16-36	Control (CONTROL)	16-57
16-37	Periodic Schedule Organization	16-60
16-38	Frame List Link Pointer Format	16-60
16-39	Asynchronous Schedule Organization	16-61
16-40	Isochronous Transaction Descriptor (iTD)	16-62
16-41	Split-Transaction Isochronous Transaction Descriptor (siTD)	16-65
16-42	Queue Element Transfer Descriptor (qTD)	16-69
16-43	Queue Head Layout	16-74
16-44	Frame Span Traversal Node Structure	16-78
16-45	Derivation of Pointer into Frame List Array	16-85
16-46	General Format of Asynchronous Schedule List	16-85
16-47	Frame Boundary Relationship between HS Bus and FS/LS Bus	16-86
16-48	Relationship of Periodic Schedule Frame Boundaries to Bus Frame Boundaries	16-87

Figures

Figure Number	Title	Page Number
16-49	Example Periodic Schedule	16-89
16-50	Example Association of iTDs to Client Request Buffer	16-92
16-51	Generic Queue Head Unlink Scenario	16-97
16-52	Asynchronous Schedule List with Annotation to Mark Head of List	16-98
16-53	Example Mapping of qTD Buffer Pointers to Buffer Pages	16-100
16-54	Host Controller Asynchronous Schedule Split-Transaction State Machine	16-104
16-55	Split Transaction, Interrupt Scheduling Boundary Conditions	16-106
16-56	General Structure of EHCI Periodic Schedule Utilizing Interrupt Spreading	16-107
16-57	Example Host Controller Traversal of Recovery Path via FSTNs	16-109
16-58	Split Transaction State Machine for Interrupt	16-113
16-59	Split Transaction, Isochronous Scheduling Boundary Conditions	16-119
16-60	siTD Scheduling Boundary Examples	16-121
16-61	Split Transaction State Machine for Isochronous	16-124
16-62	End Point Queue Head Organization	16-137
16-63	Endpoint Queue Head Layout	16-138
16-64	Endpoint Transfer Descriptor (dTD)	16-140
16-65	USB 2.0 Device States	16-144
16-66	Endpoint Queue Head Diagram	16-157
16-67	Software Link Pointers	16-159
17-1	I^2C Block Diagram	17-1
17-2	I^2C*n* Address Register (I2C*n*ADR)	17-5
17-3	I^2C*n* Frequency Divider Register (I2C*n*FDR)	17-5
17-4	I^2C*n* Control Register (I2C*n*CR)	17-6
17-5	I^2C*n* Status Register (I2C*n*SR)	17-8
17-6	I^2C*n* Data Register (I2C*n*DR)	17-9
17-7	I^2C*n* Digital Filter Sampling Rate Register (I2C*n*DFSRR)	17-10
17-8	I^2C Interface Transaction Protocol	17-11
17-9	EEPROM Contents	17-18
17-10	EEPROM Data Format for One Register Preload Command	17-19
17-11	Example I^2C Interrupt Service Routine Flowchart	17-21
18-1	UART Block Diagram	18-2
18-2	Receiver Buffer Registers (URBR1 and URBR2)	18-6
18-3	Transmitter Holding Registers (UTHR1 and UTHR2)	18-6
18-4	Divisor Most Significant Byte Registers (UDMB1 and UDMB2)	18-7
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)	18-11
18-9	Line Control Register (ULCR1 and ULCR2)	18-12
18-10	Modem Control Register (UMCR1 and UMCR2)	18-13
18-11	Line Status Register (ULSR1 and ULSR2)	18-14

Figures

Figure Number	Title	Page Number
18-12	Modem Status Register (UMSR1 and UMSR2)	18-15
18-13	Scratch Register (USCR)	18-16
18-14	Alternate Function Register (UAFR)	18-16
18-15	DMA Status Register (UDSR)	18-17
18-16	UART Bus Interface Transaction Protocol Example	18-19
19-1	SPI Block Diagram	19-2
19-2	Single-Master/Multi-Slave Configuration	19-4
19-3	Multiple-Master Configuration	19-6
19-4	SPMODE-SPI Mode Register	19-9
19-5	SPI Transfer Format with SPMODE[CP] = 0	19-11
19-6	SPI Transfer Format with SPMODE[CP] = 1	19-11
19-7	SPIE—SPI Event Register	19-12
19-8	SPIM—SPI Mask Register	19-13
19-9	SPI Command Register (SPCOM)	19-14
19-10	SPI Transmit Data Hold Register	19-14
19-11	SPI Receive Data Hold Register	19-15
19-12	Example SPMODE[REV] = 0 SPMODE[LEN] = 7 LSB Sent First	19-15
19-13	Example SPMODE[REV] = 1 SPMODE[LEN] = 7 MSB Sent First	19-15
19-14	Example SPMODE[REV] = 1 SPMODE[LEN] = 15 MSB Sent First	19-16
19-15	Example SPMODE[REV] = 0 SPMODE[LEN] = 15 LSB Sent First	19-16
20-1	JTAG Interface Block Diagram	20-1
21-1	GPIO*n* Module Block Diagram	21-1
21-2	GPIO*n* Direction Register (GP*n*DIR)	21-3
21-3	GPIO*n* Open Drain Register (GP*1*ODR–GP*2*ODR)	21-4
21-4	GPIO*n* Data Register (GP*1*DAT–GP2DAT)	21-4
21-5	GPIO*n* Interrupt Event Register (GP*1*IER–GP2IER)	21-5
21-6	GPIO*n* Interrupt Mask Register (GP1IMR–GP2IMR)	21-5
21-7	GPIO*n* Interrupt Control Register (GP*1*ICR–GP2ICR)	21-6
22-1	DLL Block Diagram	22-1
22-2	DLL Application Example	22-3
22-3	DLL Override Register	22-4
22-4	DLL Status Register	22-4
22-5	DLL Clock Register	22-5

Tables

Table Number	Title	Page Number
1-1	Functionality of the MPC8349EA, MPC8347EA, and MPC8343EA	1-1
1-2	Differences Between MPC8349E and MPC8349EA	1-18
2-1	IMMR Memory Map	2-2
2-2	Memory Map	2-3
3-1	MPC8349EA Signal Reference by Functional Block	3-4
3-2	MPC8349EA Alphabetical Signal Reference	3-11
3-3	Reset Configuration Signals	3-19
3-4	Output Signal States During System Reset	3-20
3-5	Bidirectional Signals Active During Reset	3-21
3-6	Signals for Multiplexing	3-21
4-1	System Control Signals	4-1
4-2	External Clock Signals	4-3
4-3	Reset Causes	4-4
4-4	Reset Actions	4-5
4-5	Reset Configuration Words Source	4-9
4-6	CLKIN Division	4-10
4-7	Selecting Reset Configuration Input Signals	4-10
4-8	Reset Configuration Word Low Bit Settings	4-12
4-9	System PLL Ratio	4-13
4-10	SPMF Maximum Values	4-13
4-11	Reset Configuration Word High Bit Settings	4-14
4-12	PCI Host/Agent Configuration	4-16
4-13	Boot Memory Space	4-16
4-14	Boot Sequencer Configuration	4-17
4-15	Boot ROM Location	4-18
4-16	TSEC1 Mode Configuration	4-19
4-17	TSEC2 Mode Configuration	4-20
4-18	e300 Core True Little-Endian	4-20
4-19	LALE Configuration	4-21
4-20	LDP Configuration	4-21
4-21	Local Bus Configuration EEPROM Addresses	4-21
4-22	Local Bus Reset Configuration Words Data Structure	4-22
4-23	Hard-Coded Reset Configuration Word Low Fields Values	4-27
4-24	Hard-Coded Reset Configuration Word High Field Values	4-28
4-25	Examples For Hard-Coded Reset Configuration Words Usage	4-29
4-26	Configurable Clock Units	4-31
4-27	Reset Configuration and Status Registers Memory Map	4-32
4-28	Reset Status Register Field Descriptions	4-33

Tables

Table Number	Title	Page Number
4-29	RMR Field Descriptions	4-35
4-30	RPR Bit Settings	4-35
4-31	RCR Bit Settings	4-36
4-32	RCER Bit Settings	4-37
4-33	Clock Configuration Registers Memory Map	4-37
4-34	System PLL Mode Register Bit Settings	4-38
4-35	OCCR Bit Settings	4-38
4-36	SCCR Bit Settings	4-40
4-37	Clock Control DDR Register Address Map	4-41
4-38	MCKENR Field Description	4-42
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	PCILAWBAR0–PCILAWBAR1 Bit Settings	5-10
5-12	PCILAWBAR0[BASE_ADDR] Reset Value	5-10
5-13	PCILAWAR0–PCILAWAR1 Bit Settings	5-11
5-14	PCILAWAR0[EN] Reset Value	5-11
5-15	DDRLAWBAR0–DDRLAWBAR1 Bit Settings	5-12
5-16	DDRLAWBAR0[BASE_ADDR] Reset Value	5-12
5-17	DDRLAWAR0–DDRLAWAR1 Bit Settings	5-13
5-18	DDRLAWAR0[EN] Reset Value	5-14
5-19	Overlapping Local Access Windows	5-14
5-20	System Configuration Register Memory Map	5-16
5-21	SGPRL Bit Settings	5-17
5-22	SGPRH Bit Settings	5-17
5-23	SPRIDR Bit Settings	5-18
5-24	PARTID Coding	5-18
5-25	REVID Coding	5-19
5-26	SPCR Bit Settings	5-19
5-27	SICRL Bit Settings	5-22
5-28	SICRH Bit Settings	5-25
5-29	SICRH[30–31] Bit Settings	5-28
5-30	DDRCDR Field Descriptions	5-30
5-31	DDRDSR Field Descriptions	5-31

Tables

Table Number	Title	Page Number
5-32	WDT Register Address Map	5-33
5-33	SWCRR Bit Settings	5-33
5-34	SWCNR Bit Settings	5-34
5-35	SWSRR Bit Settings	5-35
5-36	RTC Signal Properties	5-39
5-37	RTC External Signal—Detailed Signal Description	5-39
5-38	RTC Register Address Map	5-40
5-39	RTCNR Bit Settings	5-40
5-40	RTLDR Bit Settings	5-41
5-41	RTPSR Bit Settings	5-42
5-42	RTCTR Bit Settings	5-42
5-43	RTEVR Bit Settings	5-43
5-44	RTALR Bit Settings	5-43
5-45	PIT Signal Properties	5-46
5-46	PIT External Signal—Detailed Signal Descriptions	5-47
5-47	PIT Register Address Map	5-47
5-48	PTCNR Bit Settings	5-48
5-49	PTLDR Bit Settings	5-48
5-50	PTPSR Bit Settings	5-49
5-51	PTCTR Bit Settings	5-49
5-52	PTEVR Bit Settings	5-50
5-53	GTM Signal Properties	5-54
5-54	GTM External Signals—Detailed Signal Descriptions	5-55
5-55	GTM Register Address Map	5-56
5-56	GTCFR1 Bit Settings	5-58
5-57	GTCFR2 Bit Settings	5-59
5-58	GTMDR Bit Settings	5-61
5-59	GTRFR Bit Settings	5-62
5-60	GTCPR*n* Bit Settings	5-62
5-61	GTCNR Bit Settings	5-63
5-62	GTEVR*n* Bit Settings	5-64
5-63	GTPSR*n* 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	PMCER Bit Settings	5-70
5-68	PMCMR Bit Settings	5-71
6-1	Arbiter Register Map	6-2
6-2	ACR Field Descriptions	6-3
6-3	ATR Field Descriptions	6-4
6-4	AER Field Descriptions	6-5

Tables

Table Number	Title	Page Number
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	6-15
6-12	Illegal Transaction Type Encoding	6-16
7-1	MSR Bit Descriptions	7-16
7-2	e300 HID0 Bit Descriptions	7-20
7-3	HID1 Bit Descriptions	7-23
7-4	e300HID2 Bit Descriptions	7-23
7-5	Interrupt Classifications	7-31
7-6	Exceptions and Interrupts	7-31
7-7	Differences Between e300c1 and G2_LE Cores	7-38
8-1	IPIC Signal Properties	8-5
8-2	IPIC External Signals—Detailed Signal Descriptions	8-6
8-3	IPIC Register Address MapIRQ3	8-7
8-4	SICFR Field Descriptions	8-8
8-5	SIVCR Field Descriptions	8-9
8-6	IVEC/CVEC/MVEC Field Definition	8-10
8-7	SIPNR_H/SIFCR_H/SIMSR_H Bit Assignments	8-12
8-8	SIPNR_H Field Descriptions	8-12
8-9	SIPNR_L/SIFCR_L/SIMSR_L Bit Assignments	8-13
8-10	SIPNR_L Field Descriptions	8-13
8-11	SIPRR_A Field Descriptions	8-14
8-12	SIPRR_D Field Descriptions	8-15
8-13	SIMSR_H Field Descriptions	8-16
8-14	SIMSR_L Field Descriptions	8-16
8-15	SICNR Field Descriptions	8-17
8-16	SEPNR Field Descriptions	8-18
8-17	SMPRR_A Field Descriptions	8-19
8-18	SMPRR_B Field Descriptions	8-20
8-19	SEMSR Field Descriptions	8-21
8-20	SECNR Field Descriptions	8-22
8-21	SERSR/SERMR/SERFR Bit Assignments	8-23
8-22	SERSR Field Descriptions	8-23
8-23	SERMR Field Descriptions	8-24
8-24	SERCR Field Descriptions	8-24
8-25	SIFCR_H Field Descriptions	8-25
8-26	SIFCR_L Field Descriptions	8-25

Tables

Table Number	Title	Page Number
8-27	SEFCR Field Descriptions	8-26
8-28	SERFR Field Descriptions	8-27
8-29	SCVCR Field Descriptions	8-27
8-30	SMVCR Field Descriptions	8-28
8-31	Interrupt Source Priority Levels	8-31
9-1	DDR Memory Interface Signal Summary	9-4
9-2	Memory Address Signal Mappings	9-4
9-3	Memory Interface Signals—Detailed Signal Descriptions	9-5
9-4	Clock Signals—Detailed Signal Descriptions	9-8
9-5	DDR Memory Controller Memory Map	9-9
9-6	CS*n*_BNDS Field Descriptions	9-11
9-7	CS*n*_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	9-25
9-16	DDR_SDRAM_MD_CNTL Field Descriptions	9-26
9-17	Settings of DDR_SDRAM_MD_CNTL Fields	9-27
9-18	DDR_SDRAM_INTERVAL Field Descriptions	9-28
9-19	DDR_DATA_INIT Field Descriptions	9-28
9-20	DDR_SDRAM_CLK_CNTL Field Descriptions	9-29
9-21	DDR_INIT_ADDR Field Descriptions	9-29
9-22	DDR_IP_REV1 Field Descriptions	9-30
9-23	DDR_IP_REV2 Field Descriptions	9-30
9-24	DATA_ERR_INJECT_HI Field Descriptions	9-31
9-25	DATA_ERR_INJECT_LO Field Descriptions	9-31
9-26	ERR_INJECT Field Descriptions	9-32
9-27	CAPTURE_DATA_HI Field Descriptions	9-32
9-28	CAPTURE_DATA_LO Field Descriptions	9-33
9-29	CAPTURE_ECC Field Descriptions	9-33
9-30	ERR_DETECT Field Descriptions	9-34
9-31	ERR_DISABLE Field Descriptions	9-35
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	9-38
9-36	Byte Lane to Data Relationship	9-43

Tables

Table Number	Title	Page Number
9-37	Supported DDR1 SDRAM Device Configurations	9-44
9-38	Supported DDR2 SDRAM Device Configurations	9-44
9-39	DDR1 Address Multiplexing for 64-bit Data Bus with Interleaving Disabled	9-45
9-40	DDR1 Address Multiplexing for 32-bit Data Bus with Interleaving Disabled	9-46
9-41	DDR2 Address Multiplexing for 64-bit Data Bus with Interleaving Disabled	9-47
9-42	DDR2 Address Multiplexing for 32-bit Data Bus with Interleaving Disabled	9-47
9-43	Example of Address Multiplexing for 64-bit Data Bus Interleaving Between Two Banks	9-48
9-44	Example of Address Multiplexing for 64-bit Data Bus Interleaving Between Four Banks	9-49
9-45	DDR SDRAM Command Table	9-51
9-46	DDR SDRAM Interface Timing Intervals	9-51
9-47	DDR SDRAM Power-Saving Modes Refresh Configuration	9-59
9-48	Memory Controller–Data Beat Ordering	9-61
9-49	DDR SDRAM ECC Syndrome Encoding	9-63
9-50	DDR SDRAM ECC Syndrome Encoding (Check Bits)	9-64
9-51	Memory Controller Errors	9-65
9-52	Memory Interface Configuration Register Initialization Parameters	9-65
9-53	Programming Differences Between Memory Types	9-66
10-1	Signal Properties—Summary	10-4
10-2	Local Bus Controller Detailed Signal Descriptions	10-5
10-3	Local Bus Controller Memory Map	10-9
10-4	BR*n* Field Descriptions	10-11
10-5	Memory Bank Sizes in Relation to Address Mask	10-13
10-6	OR*n*—GPCM Field Descriptions	10-14
10-7	OR*n*—UPM Field Descriptions	10-16
10-8	OR*n*—SDRAM Field Descriptions	10-17
10-9	MAR Field Descriptions	10-18
10-10	M*n*MR Field Descriptions	10-19
10-11	MRTPR Field Descriptions	10-22
10-12	MDR Field Description	10-22
10-13	LSDMR Field Descriptions	10-22
10-14	LURT Field Descriptions	10-25
10-15	LSRT Field Descriptions	10-25
10-16	LTESR Field Descriptions	10-26
10-17	LTEDR Field Descriptions	10-27
10-18	LTEIR Field Descriptions	10-28
10-19	LTEATR Field Descriptions	10-28
10-20	LTEAR Field Descriptions	10-29
10-21	LBCR Field Descriptions	10-30
10-22	LCRR Field Descriptions	10-31

Tables

Table Number	Title	Page Number
10-23	GPCM Write Control Signal Timing for LCRR[CLKDIV] = 4 or 8	10-37
10-24	GPCM Read Control Signal Timing for LCRR[CLKDIV] = 4 or 8	10-38
10-25	GPCM Write Control Signal Timing for LCRR[CLKDIV] = 2	10-39
10-26	GPCM Read Control Signal Timing for LCRR[CLKDIV] = 2	10-40
10-27	Boot Bank Field Values After Reset	10-47
10-28	SDRAM Interface Commands	10-50
10-29	UPM Routines Start Addresses	10-61
10-30	UPM RAM Word Field Descriptions	10-66
10-31	M*n*MR Loop Field Use	10-71
10-32	UPM Address Multiplexing	10-72
10-33	Data Bus Requirements For Read Cycle	10-85
10-34	Micron SDRAM Devices	10-87
10-35	LAD*n* Signal Connections to 128-Mbyte SDRAM	10-89
10-36	Logical Address Bus Partitioning	10-90
10-37	SDRAM Device Address Port During Address Phase	10-90
10-38	SDRAM Device Address Port During READ/WRITE Command	10-90
10-39	Register Settings for 128-Mbytes SDRAMs	10-91
10-40	Logical Address Partitioning	10-91
10-41	SDRAM Device Address Port During Address Phase	10-92
10-42	SDRAM Device Address Port During READ/WRITE Command	10-92
10-43	Register Settings for 512-Mbyte SDRAMs	10-92
10-44	UPM Synchronization Cycles	10-100
11-1	Sequencer Memory Map	11-2
11-2	POTAR*n* Field Descriptions	11-3
11-3	POBAR*n* Field Descriptions	11-4
11-4	POCMRn Field Descriptions	11-4
11-5	PMCR Field Descriptions	11-5
11-6	DTCR Field Descriptions	11-6
12-1	DMA Interface Signals—Detailed Signal Descriptions	12-2
12-2	Module Memory Map	12-3
12-3	OMISR Field Descriptions	12-5
12-4	OMIMR Field Descriptions	12-6
12-5	IMR0 and IMR1 Field Descriptions	12-7
12-6	OMR0 and OMR1 Field Descriptions	12-7
12-7	ODR Field Descriptions	12-8
12-8	IDR Field Descriptions	12-9
12-9	IMISR Field Descriptions	12-10
12-10	IMIMR Field Descriptions	12-11
12-11	DMAMRn Field Descriptions	12-12
12-12	DMASR*n* Field Descriptions	12-14
12-13	DMACDAR*n* Field Descriptions	12-15

Tables

Table Number	Title	Page Number
12-14	DMASAR*n* Field Descriptions	12-16
12-15	DMASAR*n* Field Descriptions	12-16
12-16	DMABCR*n* Field Descriptions	12-17
12-17	DMANDAR*n* Field Descriptions	12-18
12-18	DMA Segment Descriptor Fields	12-22
13-1	PCI Controller Modes	13-3
13-2	Signal Properties	13-4
13-3	PCI1 and PCI2 Interface Signals—Detailed Signal Descriptions	13-6
13-4	PCI Configuration Access Registers	13-14
13-5	PCI Memory-Mapped Registers	13-15
13-6	PCI_CONFIG_ADDRESS Field Descriptions	13-17
13-7	PCI_CONFIG_DATA Field Descriptions	13-18
13-8	PCI_ESR Field Descriptions	13-19
13-9	PCI_ECDR Field Descriptions	13-20
13-10	PCI_EER Field Descriptions	13-21
13-11	PCI_EATCR Field Descriptions	13-21
13-12	PCI_EACR Field Description	13-23
13-13	PCI_EEACR Field Description	13-23
13-14	PCI_EDLCR Field Description	13-24
13-15	PCI_EDLCR Field Description	13-24
13-16	PCI_GCR Field Descriptions	13-25
13-17	PCI_ECR Field Descriptions	13-26
13-18	PCI_GSR Field Descriptions	13-26
13-19	PITAR*n* Field Descriptions	13-27
13-20	PIBARn Field Descriptions	13-27
13-21	PIEBARn Field Descriptions	13-28
13-22	PIWARn Field Descriptions	13-28
13-23	PCI Configuration Space Registers	13-30
13-24	Vendor ID Configuration Register Field Descriptions	13-31
13-25	Device ID Configuration Register Field Descriptions	13-31
13-26	PCI Command Configuration Register Field Descriptions	13-32
13-27	PCI Status Configuration Register Field Descriptions	13-33
13-28	Revision ID Configuration Register Field Descriptions	13-34
13-29	Standard Programming Interface Configuration Register Field Descriptions	13-34
13-30	Subclass Code Configuration Register Field Descriptions	13-35
13-31	Class Code Configuration Register Field Descriptions	13-35
13-32	Cache Line Size Configuration Register Field Descriptions	13-36
13-33	Latency Timer Configuration Register Field Descriptions	13-36
13-34	PIMMR Base Address Configuration Register Field Descriptions	13-37
13-35	GPL Base Address Register 0 Field Descriptions	13-38
13-36	GPL Base Address Register 1,2 Field Descriptions	13-39

Tables

Table Number	Title	Page Number
13-37	GPL Extended Base Address Registers 1–2 Field Descriptions	13-39
13-38	Subsystem Vendor ID Configuration Register Field Descriptions	13-40
13-39	Subsystem Device ID Configuration Register Field Descriptions	13-40
13-40	Interrupt Line Configuration Register Field Descriptions	13-41
13-41	PCI Function Configuration Register Field Descriptions	13-42
13-42	PCI Arbiter Control Register (PCIACR) Field Descriptions	13-43
13-43	Hot Swap Register Block Field Descriptions	13-44
13-44	PCI Command Definitions	13-47
13-45	Special Cycle Commands	13-57
14-1	Example Data Packet Descriptor	14-4
14-2	SEC Base Address Map	14-10
14-3	SEC Address Map	14-11
14-4	Header Dword Bit Definitions	14-17
14-5	EU_SEL1 and EU_SEL2 Values	14-18
14-6	Descriptor Types	14-18
14-7	Pointer Dword Field Definitions	14-20
14-8	Link Table Field Definitions	14-21
14-9	Descriptor Pointer Dword Usage	14-25
14-10	Mode Field Description	14-27
14-11	PKEURCR Field Descriptions	14-30
14-12	PKEU Status Register Field Descriptions	14-30
14-13	PKEUISR Field Descriptions	14-31
14-14	PKEUICR Field Descriptions	14-33
14-15	DEUMR Field Descriptions	14-35
14-16	DEUKSR Field Descriptions	14-36
14-17	DEURCR Field Descriptions	14-37
14-18	DEUSR Field Descriptions	14-38
14-19	DEUISR Field Descriptions	14-39
14-20	DEUICR Field Descriptions	14-40
14-21	AFEUMR Field Descriptions	14-44
14-22	AFEURCR Field Descriptions	14-46
14-23	AFEUSR Field Descriptions	14-47
14-24	AFEUISR Field Descriptions	14-48
14-25	AFEUICR Field Descriptions	14-49
14-26	MDEUMR in ‘Old’ Configuration	14-52
14-27	MDEUMR in ‘New’ Configuration	14-53
14-28	Mode Register—HMAC or SSL-MAC Generated by Single Descriptor	14-55
14-29	Mode Register—HMAC Generated across a Sequence of Descriptors	14-55
14-30	MDEU Reset Control Register Field Descriptions	14-56
14-31	MDEU Status Register Field Descriptions	14-57
14-32	MDEUISR Field Descriptions	14-58

Tables

Table Number	Title	Page Number
14-33	MDEUICR Field Descriptions	14-60
14-34	RNG Mode Register Definitions	14-64
14-35	RNG Reset Control Register Field Descriptions	14-65
14-36	RNG Status Register Field Descriptions	14-66
14-37	RNG Interrupt Status Register Field Descriptions	14-66
14-38	RNG Interrupt Control Register Field Descriptions	14-67
14-39	AESU Mode Register Field Descriptions	14-69
14-40	AES Cipher Modes	14-70
14-41	AESU Reset Control Register Field Descriptions	14-72
14-42	AESU Status Register Field Descriptions	14-73
14-43	AESU Interrupt Status Register Field Descriptions	14-74
14-44	AESU Interrupt Control Register Field Descriptions	14-75
14-45	Counter Modulus	14-78
14-46	CCCR Field Descriptions	14-83
14-47	Header Dword Writeback Field Descriptions	14-84
14-48	Crypto-Channel Pointer Status Register Signals	14-85
14-49	G_STATE Field Values	14-87
14-50	S_STATE Field Values	14-87
14-51	CHN_STATE Field Values	14-88
14-52	Crypto-Channel Pointer Status Register Error Field Definitions	14-89
14-53	Channel Pointer Status Register PTR_DW Field Values	14-89
14-54	Crypto-Channel Current Descriptor Pointer Register Signals	14-90
14-55	Fetch FIFO Field Descriptions	14-91
14-56	Channel Assignment Value	14-94
14-57	Interrupt Mask, Status, and Clear Register Fields	14-97
14-58	Master Control Register Signals	14-99
15-1	TSEC Signals—Detailed Signal Descriptions	15-9
15-2	Module Memory Map Summary	15-13
15-3	Module Memory Map	15-13
15-4	IEVENT Field Descriptions	15-20
15-5	IMASK Field Descriptions	15-23
15-6	EDIS Field Descriptions	15-24
15-7	ECNTRL Field Descriptions	15-25
15-8	MINFLR Field Descriptions	15-27
15-9	PTV Field Descriptions	15-27
15-10	DMACTRL Field Descriptions	15-28
15-11	TBIPA Field Descriptions	15-30
15-12	FIFO_TX_THR Field Descriptions	15-31
15-13	FIFO_TX_SP Field Descriptions	15-31
15-14	FIFO_TX_STARVE Field Descriptions	15-32
15-15	FIFO_TX_STARVE_SHUTOFF Field Descriptions	15-32

Tables

Table Number	Title	Page Number
15-16	TCTRL Field Descriptions	15-33
15-17	TSTAT Field Descriptions	15-34
15-18	TBDLEN Field Descriptions	15-34
15-19	TXIC Field Descriptions	15-35
15-20	CTBPTR Field Descriptions	15-36
15-21	TBPTR Field Descriptions	15-37
15-22	TBASE Field Descriptions	15-37
15-23	OSTBD Field Descriptions	15-38
15-24	OSTBDP Field Descriptions	15-40
15-25	RCTRL Field Descriptions	15-40
15-26	RSTAT Field Descriptions	15-41
15-27	RBDLEN Field Descriptions	15-42
15-28	RXIC Field Descriptions	15-42
15-29	CRBPTR Field Descriptions	15-43
15-30	MRBLR Field Descriptions	15-44
15-31	RBPTR Field Descriptions	15-44
15-32	RBASE Field Descriptions	15-45
15-33	MACCFG1 Field Descriptions	15-48
15-34	MACCFG2 Field Descriptions	15-50
15-35	IPGIFG Field Descriptions	15-51
15-36	HAFDUP Field Descriptions	15-52
15-37	MAXFRM Field Descriptions	15-53
15-38	MIIMCFG Field Descriptions	15-54
15-39	MIIMCOM Field Descriptions	15-54
15-40	MIIMADD Field Descriptions	15-55
15-41	MIIMCON Field Descriptions	15-56
15-42	MIIMSTAT Field Descriptions	15-56
15-43	MIIMIND Field Descriptions	15-57
15-44	IFSTAT Field Descriptions	15-58
15-45	MACSTNADDR1 Field Descriptions	15-59
15-46	MACSTNADDR2 Field Descriptions	15-59
15-47	TR64 Field Descriptions	15-60
15-48	TR127 Field Descriptions	15-61
15-49	TR255 Field Descriptions	15-61
15-50	TR511 Field Descriptions	15-62
15-51	TR1K Field Descriptions	15-62
15-52	TRMAX Field Descriptions	15-63
15-53	TRMGV Field Descriptions	15-63
15-54	RBYT Field Descriptions	15-64
15-55	RPKT Field Descriptions	15-64
15-56	RFCS Field Descriptions	15-65

Tables

Table Number	Title	Page Number
15-57	RMCA Field Descriptions	15-65
15-58	RBCA Field Descriptions	15-66
15-59	RXCF Field Descriptions	15-66
15-60	RXPF Field Descriptions	15-67
15-61	RXUO Field Descriptions	15-67
15-62	RALN Field Descriptions	15-68
15-63	RFLR Field Descriptions	15-68
15-64	RCDE Field Descriptions	15-69
15-65	RCSE Field Descriptions	15-69
15-66	RUND Field Descriptions	15-70
15-67	ROVR Field Descriptions	15-70
15-68	RFRG Field Descriptions	15-71
15-69	RJBR Field Descriptions	15-71
15-70	RDRP Field Descriptions	15-72
15-71	TBYT Field Descriptions	15-72
15-72	TPKT Field Descriptions	15-73
15-73	TMCA Field Descriptions	15-73
15-74	TBCA Field Descriptions	15-74
15-75	TXPF Field Descriptions	15-74
15-76	TDFR Field Descriptions	15-75
15-77	TEDF Field Descriptions	15-75
15-78	TSCL Field Descriptions	15-76
15-79	TMCL Field Descriptions	15-76
15-80	TLCL Field Descriptions	15-77
15-81	TXCL Field Descriptions	15-77
15-82	TNCL Field Descriptions	15-78
15-83	TDRP Field Descriptions	15-78
15-84	TJBR Field Descriptions	15-79
15-85	TFCS Field Descriptions	15-79
15-86	TXCF Field Descriptions	15-80
15-87	TOVR Field Descriptions	15-80
15-88	TUND Field Descriptions	15-81
15-89	TFRG Field Descriptions	15-81
15-90	CAR1 Field Descriptions	15-82
15-91	CAR2 Field Descriptions	15-84
15-92	CAM1 Field Descriptions	15-85
15-93	CAM2 Field Descriptions	15-87
15-94	IADDR*n* Field Description	15-88
15-95	GADDR*n* Field Description	15-89
15-96	ATTR Field Descriptions	15-89
15-97	TBI MII Register Set	15-90

Tables

Table Number	Title	Page Number
15-98	CR Field Descriptions	15-91
15-99	SR Descriptions	15-92
15-100	ANA Field Descriptions	15-93
15-101	PAUSE Priority Resolution	15-94
15-102	ANLPBPA Field Descriptions	15-95
15-103	ANEX Field Descriptions	15-96
15-104	ANNPT Field Descriptions	15-97
15-105	ANLPANP Field Descriptions	15-98
15-106	EXST Field Descriptions	15-99
15-107	JD Field Description	15-100
15-108	TBICON Field Description	15-101
15-109	GMII, MII, and TBI Signal Multiplexing	15-106
15-110	RGMII and RTBI Signal Multiplexing	15-107
15-111	Shared Signals	15-109
15-112	Steps of Minimum Register Initialization	15-109
15-113	Flow Control Frame Structure	15-116
15-114	Non-Error Transmit Interrupts	15-117
15-115	Non-Error Receive Interrupts	15-117
15-116	Interrupt Coalescing Timing Threshold Ranges	15-118
15-117	Transmission Errors	15-119
15-118	Reception Errors	15-120
15-119	Transmit Data Buffer Descriptor (TxBD) Field Descriptions	15-122
15-120	Receive Buffer Descriptor Field Descriptions	15-124
15-121	MII Interface Mode Signal Configuration	15-126
15-122	Shared MII Signals	15-127
15-123	MII Mode Register Initialization Steps	15-127
15-124	GMII Interface Mode Signal Configuration	15-130
15-125	Shared GMII Signals	15-131
15-126	GMII Mode Register Initialization Steps	15-131
15-127	TBI Interface Mode Signal Configuration	15-133
15-128	Shared TBI Signals	15-134
15-129	TBI Mode Register Initialization Steps	15-135
15-130	RGMII Interface Mode Signal Configuration	15-137
15-131	Shared RGMII Signals	15-138
15-132	RGMII Mode Register Initialization Steps	15-139
15-133	RTBI Interlace Mode Signal Configuration	15-141
15-134	Shared RTBI Signals	15-142
15-135	RTBI Mode Register Initialization Steps	15-142
16-1	External Port Configurations	16-4
16-2	USB External Signals	16-4
16-3	ULPI Signal Descriptions	16-7

Tables

Table Number	Title	Page Number
16-4	UTMI Signal Descriptions	16-9
16-5	Serial Detailed Signal Descriptions	16-11
16-6	Serial OTG Detailed Signal Descriptions	16-13
16-7	USB Interface Memory Map	16-16
16-8	CAPLENGTH Register Field Descriptions	16-19
16-9	HCIVERSION Register Field Descriptions	16-19
16-10	HCSPARAMS Register Field Descriptions	16-20
16-11	HCCPARAMS Register Field Descriptions	16-21
16-12	DCIVERSION Register Field Descriptions	16-22
16-13	DCCPARAMS Register Field Descriptions	16-22
16-14	USBCMD Register Field Descriptions	16-23
16-15	USBSTS Register Field Descriptions	16-26
16-16	USBINTR Register Field Descriptions	16-29
16-17	FRINDEX Register Field Descriptions	16-31
16-18	FRINDEX N Values	16-31
16-19	PERIODICLISTBASE Register Field Descriptions	16-32
16-20	DEVICEADDR Register Field Descriptions	16-32
16-21	ASYNCLISTADDR Register Field Descriptions	16-33
16-22	ENDPOINTLISTADDR Register Field Descriptions	16-33
16-23	ASYNCTTSTS Register Field Descriptions	16-34
16-24	BURSTSIZE Register Field Descriptions	16-35
16-25	TXFILLTUNING Register Field Descriptions	16-36
16-26	TXTTFILLTUNING Register Field Description	16-37
16-27	ULPI VIEWPORT Field Descriptions	16-38
16-28	CONFIGFLAG Register Field Descriptions	16-39
16-29	PORTSC*n* Register Field Descriptions	16-40
16-30	OTGSC Register Field Descriptions	16-45
16-31	USBMODE Register Field Descriptions	16-47
16-32	ENDPTSETUPSTAT Register Field Descriptions	16-48
16-33	ENDPTPRIME Register Field Descriptions	16-48
16-34	ENDPTFLUSH Register Field Descriptions	16-49
16-35	ENDPTSTATUS Register Field Descriptions	16-50
16-36	ENDPTCOMPLETE Register Field Descriptions	16-51
16-37	ENDPTCTRL0 Register Field Descriptions	16-51
16-38	ENDPTCTRLx Register Field Descriptions	16-52
16-39	SNOOP*n* Register Field Descriptions	16-54
16-40	AGE_CNT_THRESH Register Field Descriptions	16-55
16-41	SI_CTRL Register Field Descriptions	16-56
16-42	PRI_CTRL Register Field Descriptions	16-57
16-43	CONTROL Field Descriptions	16-57
16-44	Typ Field Encodings	16-61

Tables

Table Number	Title	Page Number
16-45	Next Schedule Element Pointer	16-62
16-46	iTD Transaction Status and Control	16-63
16-47	Buffer Pointer Page 0 (Plus)	16-64
16-48	iTD Buffer Pointer Page 1 (Plus)	16-64
16-49	Buffer Pointer Page 2 (Plus)	16-64
16-50	Buffer Pointer Page 3-6	16-65
16-51	Next Link Pointer	16-65
16-52	Endpoint and Transaction Translator Characteristics	16-66
16-53	Micro-frame Schedule Control	16-66
16-54	siTD Transfer Status and Control	16-67
16-55	siTD Buffer Pointer Page 0 (Plus)	16-68
16-56	siTD Buffer Pointer Page 1 (Plus)	16-68
16-57	siTD Back Link Pointer	16-68
16-58	qTD Next Element Transfer Pointer (DWord 0)	16-69
16-59	qTD Alternate Next Element Transfer Pointer (DWord 1)	16-70
16-60	qTD Token (DWord 2)	16-70
16-61	qTD Buffer Pointer	16-73
16-62	Queue Head DWord 0	16-74
16-63	Endpoint Characteristics: Queue Head DWord 1	16-75
16-64	Endpoint Capabilities: Queue Head DWord 2	16-76
16-65	Current qTD Link Pointer	16-77
16-66	Host-Controller Rules for Bits in Overlay (DWords 5, 6, 8 and 9)	16-78
16-67	FTSN Normal Path Pointer	16-79
16-68	FSTN Back Path Link Pointer	16-79
16-69	Default Values of Operational Register Space	16-80
16-70	Behavior During Wake-up Events	16-83
16-71	Operation of FRINDEX and SOFV (SOF Value Register)	16-88
16-72	Example Periodic Reference Patterns for Interrupt Transfers	16-101
16-73	Ping Control State Transition Table	16-102
16-74	Interrupt IN/OUT Do Complete Split State Execution Criteria	16-116
16-75	Initial Conditions for OUT siTD's TP and T-count Fields	16-125
16-76	Transaction Position (TP)/Transaction Count (T-Count) Transition Table	16-125
16-77	Summary siTD Split Transaction State	16-128
16-78	Example Case 2a - Software Scheduling siTDs for an IN Endpoint	16-130
16-79	Summary of Transaction Errors	16-133
16-80	Summary Behavior on Host System Errors	16-136
16-81	Endpoint Capabilities/Characteristics	16-138
16-82	Current dTD Pointer	16-139
16-83	Multiple Mode Control	16-140
16-84	Next dTD Pointer	16-140
16-85	dTD Token	16-141

Tables

Table Number	Title	Page Number
16-86	Buffer Pointer Page 0	16-142
16-87	Buffer Pointer Page 1	16-142
16-88	Buffer Pointer Pages 2–4	16-142
16-89	Device Controller State Information Bits	16-145
16-90	Device Controller Endpoint Initialization	16-148
16-91	Device Controller Stall Response Matrix	16-148
16-92	Variable Length Transfer Protocol Example (ZLT=0)	16-150
16-93	Variable Length Transfer Protocol Example (ZLT=1)	16-151
16-94	Interrupt/Bulk Endpoint Bus Response Matrix	16-152
16-95	Control Endpoint Bus Response Matrix	16-154
16-96	Isochronous Endpoint Bus Response Matrix	16-156
16-97	Device Error Matrix	16-161
16-98	Error Descriptions	16-162
16-99	Interrupt Handling Order	16-162
16-100	Low Frequency Interrupt Events	16-162
16-101	Error Interrupt Events	16-163
16-102	Functional Differences Between EHCI and EHCI with Embedded TT	16-164
16-103	Emulated Handshakes	16-166
17-1	I^2C Interface Signal Description	17-3
17-2	I^2C Interface Signals—Detailed Signal Descriptions	17-4
17-3	I^2C Memory Map	17-4
17-4	I2C*n*ADR Field Descriptions	17-5
17-5	I2C*n* FDR Field Descriptions	17-6
17-6	I2C*n*CR Field Descriptions	17-7
17-7	I2C*n*SR Field Descriptions	17-8
17-8	I2C*n*DR Field Description	17-9
17-9	I2C*n*DFSRR 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	18-5
18-4	URBR Field Description	18-6
18-5	UTHR Field Description	18-7
18-6	UDMB Field Description	18-7
18-7	UDLB Field Description	18-7
18-8	Baud Rate Examples	18-8
18-9	UIER Field Descriptions	18-9
18-10	UIIR Field Descriptions	18-10
18-11	UIIR IID Bits Summary	18-10
18-12	UFCR Field Descriptions	18-11
18-13	ULCR Field Descriptions	18-12
18-14	Parity Selection Using ULCR[PEN], ULCR[SP], and ULCR[EPS]	18-13

Tables

Table Number	Title	Page Number
18-15	UMCR Field Descriptions	18-13
18-16	ULSR Field Descriptions	18-14
18-17	UMSR Field Descriptions	18-15
18-18	USCR Field Descriptions	18-16
18-19	UAFR Field Descriptions	18-16
18-20	UDSR Field Descriptions	18-17
18-21	UDSR[TXRDY] Set Conditions	18-17
18-22	UDSR[TXRDY] Cleared Conditions	18-17
18-23	UDSR[RXRDY] Set Conditions	18-18
18-24	UDSR[RXRDY] Cleared	18-18
19-1	Signal Properties	19-7
19-2	Detailed Signal Descriptions	19-7
19-3	SPI Register Summary	19-9
19-4	SPMODE Field Descriptions	19-9
19-5	SPIE Field Descriptions	19-12
19-6	SPIM Field Descriptions	19-13
19-7	SPCOM Field Descriptions	19-14
19-8	SPI Transmit Data Hold Register	19-14
19-9	SPI Receive Data Hold Register	19-15
20-1	JTAG Test Signals Summary	20-2
20-2	JTAG Test—Detailed Signal Descriptions	20-2
21-1	GPIO External Signals—Detailed Signal Descriptions	21-2
21-2	GPIO Register Address Map	21-2
21-3	GP*n*DIR Bit Settings	21-3
21-4	GP*n*ODR Bit Settings	21-4
21-5	GP*n*DAT Bit Settings	21-4
21-6	GP*n*IER Bit Settings	21-5
21-7	GP*n*IMR Bit Settings	21-5
21-8	GP*n*ICR Bit Settings	21-6
22-1	DLL Macro External Signals	22-2
22-2	DLL Register Address Map	22-3
22-3	DLLOVR Field Description	22-4
22-4	DLLSR Field Description	22-5
22-5	DLLCK Field Description	22-5

Tables

Table Number	Title	Page Number

About This Book

This reference manual defines the functionality of the MPC8349EA, MPC8347EA, and MPC8343EA. It is written from the perspective a user of the MPC8349EA, which is the superset device.

The MPC8349EA is a next-generation PowerQUICC II™ Pro integrated host processor. It provides integration of processing power for networking infrastructure, telecommunications, wireless LANs, and other embedded applications, resulting in higher device performance. The MPC8349EA contains an embedded PowerPC™ core, which is built on Power Architecture™ technology. The e300 processor core, with its 32 Kbytes of instruction and data cache, implements the PowerPC user instruction set architecture and provides hardware and software debugging support.

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 MPC8349EA integrated host processor. 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, “Memory Map,” describes the device memory map, starting with an overview of the local address map. Next, a complete listing of all memory-mapped registers is provided, with cross references to the sections detailing descriptions of each.
- Chapter 3, “Signal Descriptions,” provides a listing of all the external signals, cross-references for signals that serve multiple functions, their functional blocks, and I/O states. These signals are also listed in alphabetical order.
- 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. It also describes 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 DDR SDRAM memory controller of the MPC8349EA. This fully programmable controller supports most DDR memories available today, including both buffered and unbuffered devices. The built-in error checking and correction (ECC) ensures very low bit-error rates for reliable high-frequency operation. Dynamic power management and auto-precharge modes simplify memory system design. Special features like ECC error injection support rapid system debug.
- Chapter 10, “Local Bus Controller,” describes the local bus controller (LBC). It describes the external signals and the memory-mapped registers as well as a functional description of the general-purpose chip-select machine (GPCM), synchronous DRAM (SDRAM) machine, and user-programmable machines (UPMs) of the LBC. Also, it includes an initialization and applications information section with many specific examples of its use.
- Chapter 11, “Sequencer,” describes how the I/O sequencer switches transactions among its ports, using a buffer pool to minimize blocking. It also provides address translation on outbound PCI transactions.
- Chapter 12, “DMA/Messaging Unit,” describes the four-channel high-speed general-purpose DMA controller of the MPC8349EA. The channels share buffer space in the IOS to facilitate the gathering and sending of data. The DMA/messaging unit supports communication between two processors on different buses, for example, a local processor and a processor on a PCI bus. This communication unit operates with generic messages and doorbell registers. This block also provides a DMA controller that transfers blocks of data independent of the local processor or PCI hosts.
- Chapter 13, “PCI Bus Interface,” describes the PCI interface, which complies with the *PCI Local Bus Specification*, Rev. 2.3. This chapter provides a basic description of PCI bus operations. The specific emphasis is directed at how this device implements the PCI specification.
- Chapter 14, “Security Engine (SEC) 2.4,” describes the SEC 2.0, designed to offload computationally intensive security functions, such as key generation and exchange, authentication, and bulk encryption, from the e300 core. It is optimized to process all algorithms associated with IPSec, IKE, SSL/TLS, iSCSI, SRTP, and 802.11i. The SEC 2.0 is derived from integrated security cores found in other members of the PowerQUICC family, including SEC 1.0, the version implemented in the MPC8272/MPC8248.
- Chapter 15, “Three-Speed Ethernet Controllers,” describes the two three-speed Ethernet controllers on the MPC8349EA. The 10/100 Ethernet functionality provides Ethernet speeds from 10 to 100 megabits per second (Mbps) and provides a simple, cost-effective option for backbone and server connectivity. This three-speed Ethernet controller (TSEC) also implements a gigabit Ethernet protocol, which builds upon the 10/100 Ethernet protocol, with a throughput of one gigabit per second (Gbps). Each controller provides very high throughput using a captive DMA channel and direct connection to the MPC8349EA memory coherency module.

- Chapter 16, “Universal Serial Bus Interface,” describes the universal serial bus (USB) interface. The device implements two USB modules, a multi-port host (MPH) module and a dual-role (DR) module. These modules may be connected to one or two external ports. Both the MPH and DR modules are USB 2.0-compliant serial interface engines for implementing a USB interface. The registers and data structures for both modules are based on the *Enhanced Host Controller Interface Specification for Universal Serial Bus* (EHCI) from Intel Corporation. The MPH module can act as a host controller for either one or two USB ports. The DR module can act as a host, as a device, or as an on-the-go (OTG) negotiable host/device on the USB bus.
- Chapter 17, “I^2C Interface,” describes the inter-IC (IIC or I^2C) bus controller. This synchronous, serial, bidirectional, multiple-master bus allows two-wire connection of devices, such as real-time clock devices, EEPROMs, microcontrollers, A/D converters, and LCDs. The MPC8349EA powers up in boot sequencer mode which allows the I^2C controller 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 serial peripheral interface (SPI) that allows the exchange of data between other PowerQUICC family chips, the MC68360, the MC68302, M68HC11 and M68HC05 microcontroller families, and other MPC83xx family devices. The SPI can also be used to communicate with peripheral devices such as EEPROMs, real-time clocks, A/D converters, and ISDN devices.
- Chapter 20, “JTAG/Testing Support,” describes the joint test action group (JTAG) interface, which facilitates 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) modules in the MPC8349EA device, including a definition of the external signals and functions they serve. Additionally, interrupt capabilities, pin description, and register settings are described.
- Chapter 22, “Delay Lock Loop (DLL),” describes the delay lock loop (DLL) module in the device. including the configuration, control, and status registers.
- This reference manual also includes a glossary, an appendix containing a revision history, and two indexes.

Suggested Reading

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

General Information

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

- *The PowerPC Architecture: A Specification for a New Family of RISC Processors*, Second Edition, by International Business Machines, Inc.

- *Computer Architecture: A Quantitative Approach*, Third Edition, by John L. Hennessy and David A. Patterson
- *Computer Organization and Design: The Hardware/Software Interface*, Second Edition, by David A. Patterson and John L. Hennessy

Related Documentation

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

- *e300 Core Reference Manual*—This book provides a more detailed description of the e300 core.
- Reference manuals (formerly called user's manuals)—These books provide details about individual implementations.
- Addenda/errata to reference or user's manuals—Because some processors have follow-on devices, an addendum may be provided that describes the additional features and functionality changes. These addenda are intended for use with the corresponding reference or user's manuals.
- Hardware specifications—Hardware specifications provide specific data regarding bus timing, signal behavior, and AC, DC, and thermal characteristics, as well as other design considerations.
- Product briefs—Each device has a product brief that provides an overview of its features.
- Application notes—These short documents address specific design issues useful to programmers and engineers working with Freescale processors.

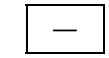
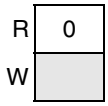
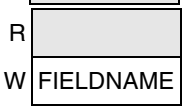
Additional literature is published as new processors become available. For a current list of documentation, refer to http://www.freescale.com.

Conventions

This document uses the following notational conventions:

cleared/set	When a bit takes the value zero, it is said to be cleared; when it takes a value of one, it is said to be set.
mnemonics	Instruction mnemonics are shown in lowercase bold.
italics	Italics indicate variable command parameters, for example, **bcctr***x*. Book titles in text are set in italics Internal signals are set in lowercase italics, for example, $\overline{core_int}$
0x0	Prefix to denote hexadecimal number
0b0	Prefix to denote binary number
rA, **r**B	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.
x	An italicized *x* indicates an alphanumeric variable.
n	An italicized *n* indicates a numeric variable.
¬	NOT logical operator
&	AND logical operator
|	OR logical operator
||	Concatenation, for example TCR[WP]||TCR[WPEXT]

	Indicates a reserved bit field in a 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. For more information, see Section 3.2, “Configuration Signals Sampled at Reset.”

Acronyms and Abbreviations

Table i contains acronyms and abbreviations used in this document.

Table i. Acronyms and Abbreviated Terms

Term	Meaning
AESU	Advanced encryption standard unit
AFEU	ARC four execution unit
BD	Buffer descriptor
BIST	Built-in self test
CD	Collision detect
COL	Collision
CPM	Communication processor module

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
CRC	Cyclic redundancy check
CRS	Carrier sense
CSB	Coherent system bus
CSMA	Carrier-sense multiple access
DDR	Double data rate
DEU	Data encryption standard execution unit
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
I^2C	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

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
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
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

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
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
ULPI	USB low-pin count interface
UPM	User-programmable machine
USB	Universal serial bus
UTMI	USB transceiver macrocell interface
UTP	Unshielded twisted pair
WDT	Watchdog timer
ZBT	Zero bus turnaround

Chapter 1
Overview

This chapter provides an overview of the MPC8349EA PowerQUICC™ II Pro processor features, including a block diagram showing the major functional components. This device is a cost-effective general-purpose integrated host processor that implements the PowerPC™ architecture required for networking infrastructure, telecommunications, wireless LANs, and other embedded applications. The processor fits applications where cost, space, power consumption, and performance are critical requirements. The device can also be used for control processing in applications such as network routers and switches, mass storage subsystems, network appliances, and print and imaging systems. Note that this reference manual covers MPC8349/47/43 devices. See Table 1-1 for the functional differences between MPC834x family devices.

1.1 MPC834x-Family Product Distinctions

Table 1-1 highlights (in bold) the primary functional differences between the MPC8349EA, MPC8347EA, and MPC8343EA.

Table 1-1. Functionality of the MPC8349EA, MPC8347EA, and MPC8343EA

Descriptions	MPC8349EA	MPC8347EA		MPC8343EA
Memory controller	64-/32-bit DDR/DDR2	64-/32-bit DDR/DDR2		**32-bit DDR/DDR2**
PCI interface	Two 32-bit or one 64-bit	**One 32-bit**		**One 32-bit**
	8 pci_clk signals	**5 pci_clk signals**		**5 pci_clk signals**
Ethernet/PHY I/F	MII, GMII, RGMII, TBI, RTBI	MII, GMII, RGMII, TBI, RTBI		**MII, RGMII, RTBI**
USB	2.0 host and device	2.0 host and device		**Either 2.0 host or 2.0 device**
	Supports two ports	Supports two ports		**Supports one port**
	MPH and DR modules	MPH and DR modules		**DR module only**
Security (DES/3DES, AES, SH1)	MPC8349EA: Yes MPC8349A: **No**	MPC8347EA: Yes MPC8347A: **No**		MPC8343EA: Yes MPC8343A: **No**
GPIO signals	64	**52**		**39**
SVR	MPC8349EA: 8050_0030 MPC8349A: 8051_0030	MPC8347EA: 8052_0030 MPC8347A: 8053_0030	MPC8347EA: 8054_0030 MPC8347A: 8055_0030	MPC8343EA: 8056_0030 MPC8343A: 8057_0030
Packaging	TBGA	TBGA	**PBGA**	**PBGA**

1.2 MPC8349EA PowerQUICC II Pro Processor Overview

Figure 1-1 shows the major functional units within the device. The PowerPC e300c1 core, with its 32 Kbytes of instruction and 32 Kbytes of data cache, implements the PowerPC user instruction set architecture and provides hardware and software debugging support. In addition, the MPC8349EA offers dual three-speed 10, 100, 1000 Mbps Ethernet controllers, a DDR/DDR2 SDRAM memory controller, a flexible local bus controller, dual 32-bit / single 64-bit PCI controllers, a dedicated security engine, a programmable interrupt controller, dual I^2C controllers, a 4-channel DMA controller, a general purpose I/O port, and a USB 2.0 host and device controller. The high level of integration in this device helps simplify board design and offers significant bandwidth and performance.

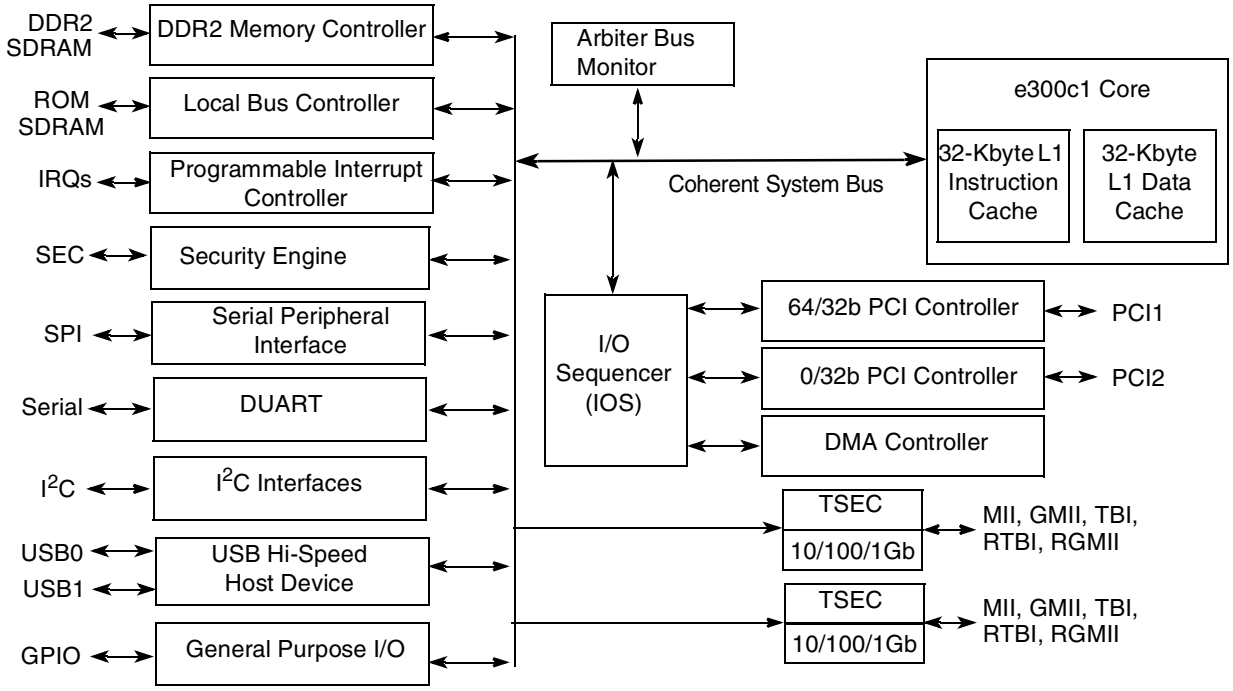


Figure 1-1. MPC8349EA Block Diagram

The major features of this device are as follows:

- e300c1 PowerPC 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, integer, load/store, system register, and branch processing units
  - 32-Kbyte instruction cache and 32-Kbyte data cache with lockable capabilities
  - Dynamic power management
  - Enhanced hardware program debug features
  - Software-compatible with Freescale processor families implementing the PowerPC architecture
  - Separate PLL that is clocked by the system bus clock

- Security engine optimized to handle all the algorithms associated with IPSec, SSL/TLS, SRTP, 802.11i, iSCSI, and IKE processing. The security engine contains four crypto-channels, a controller, and a set of crypto execution units (EUs). The execution units are:
  - Public key execution unit (PKEU) supporting the following:
    - RSA and Diffie-Hellman algorithms
    - Programmable field size up to 2048 bits
    - Elliptic curve cryptography
    - F2m and F(p) modes
    - Programmable field size up to 511 bits
  - Data encryption standard execution unit (DEU)
    - DES and 3DES algorithms
    - Two key (K1, K2) or three key (K1, K2, K3) for 3DES
    - ECB and CBC modes for both DES and 3DES
  - Advanced encryption standard unit (AESU)
    - Implements the Rijndael symmetric-key cipher
    - Key lengths of 128-, 192-, and 256-bits
    - ECB, CBC, CCM, and counter (CTR) modes
  - XOR parity generation accelerator for RAID applications
  - ARC four execution unit (AFEU)
    - Implements a stream cipher compatible with the RC4 algorithm
    - 40- to 128-bit programmable key
  - Message digest execution unit (MDEU)
    - SHA with 160-, 224-, or 256-bit message digest
    - MD5 with 128-bit message digest
    - HMAC with either algorithm
  - Random number generator (RNG)
  - Four crypto-channels, each supporting multi-command descriptor chains
    - Static and/or dynamic assignment of crypto-execution units through an integrated controller
    - Buffer size of 256 bytes for each execution unit, with flow control for large data sizes
- DDR SDRAM memory controller
  - Programmable timing supporting both DDR1 and DDR2 SDRAM
  - 32- or 64-bit data interface, up to 333-MHz data rate
  - Up to four physical banks (chip selects), each bank up to 1 Gbyte independently addressable
  - DRAM chip configurations from 64 Mbits to 1 Gbit with x8/x16/ data ports
  - Full ECC support.
  - On-die termination (ODT) support when using DDR2
  - Driver impedance calibration support (using MDIC signals) when using DDR2
  - Support for up to 16 simultaneous open pages (up to 32 pages for DDR2)

  - Contiguous or discontiguous memory mapping
  - Read-modify-write support
  - Sleep-mode support for SDRAM self refresh
  - Supports auto refresh
  - On-the-fly power management using CKE
  - Registered DIMM support
  - 2.5-V SSTL2 compatible I/O for DDR1, 1.8-V SSTL2 compatible I/O for DDR2
- Two three-speed Ethernet controllers (TSECs)
  - Three-speed support (10/100/1000 Mbps)
  - Two controllers designed to comply with IEEE Std. 802.3, 802.3u, 802.3x, 802.3z, 802.3ac
  - Support for various Ethernet physical interfaces:
    - 1000 Mbps full-duplex IEEE® Std. 802.3® GMII and IEEE Std. 802.3z TBI/RTBI/RGMII
    - 10/100 Mbps full- and half-duplex IEEE Std. 802.3 MII
  - Internal 2-Kbyte transmit and 2-Kbyte receive FIFOs per TSEC module
- Dual PCI interfaces
  - PCI specification revision 2.3 compatible
  - PCI interface(s) operating at up to 66 MHz
  - Data bus widths (2 options):
    - Dual 32-bit PCI interfaces
    - Single 64-bit PCI interface
  - PCI 3.3-V compatible
  - Not 5-V compatible
  - PCI host bridge capabilities on both interfaces
  - PCI agent mode supported on PCI1 interface
  - Support for PCI-to-memory and memory-to-PCI streaming
  - Memory prefetching of PCI read accesses and support for delayed read transactions
  - Support for posting of processor-to-PCI and PCI-to-memory writes
  - On-chip arbitration, supporting five masters on PCI1 and three masters on PCI2
  - Support for accesses to all PCI address spaces
  - Support for parity
  - Selectable hardware-enforced coherency
  - Address translation units for address mapping between host and peripheral
  - Dual address cycle support as target
  - Internal configuration registers accessible from PCI
- Universal serial bus (USB) dual role controller
  - Supports USB on-the-go mode, which includes both device and host functionality
  - Complies with USB specification revision 2.0

  - Supports operation as a stand-alone USB device
    - Supports one upstream-facing port
    - Supports six programmable bidirectional USB endpoints
  - 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 high-speed (480-Mbps), full-speed (12-Mbps), and low-speed (1.5-Mbps) operations. Low speed is only supported in host mode.
  - Supports external PHY with UTMI, serial and UTMI+ low-pin interface (ULPI)
- USB multi-port host controller
  - Supports operation as a stand-alone USB host controller
    - Supports USB root hub with one or two downstream-facing ports
    - Enhanced host controller interface (EHCI) compatible
  - Complies with USB Specification Revision 2.0
  - Supports high-speed (480 Mbps), full-speed (12 Mbps), and low-speed (1.5 Mbps) operations
  - Supports a direct connection to a high-speed device without an external hub
  - Supports external PHY with serial and low-pin count (ULPI) interfaces
- Local bus controller (LBC)
  - Multiplexed 32-bit address and data operating at up to 133 MHz
  - Eight chip selects support eight external slaves
  - Up to eight-beat burst transfers
  - 32-, 16-, and 8-bit ports are controlled by an on-chip memory controller
  - Three protocol engines available on a per chip select basis:
    - General-purpose chip select machine (GPCM)
    - Three user programmable machines (UPMs)
    - Dedicated single data rate SDRAM controller
  - Parity support
  - Default boot ROM chip select with configurable bus width (8, 16, or 32 bits)
- Integrated Programmable interrupt controller (IPIC)
  - Functional and programming compatibility with the MPC8260 interrupt controller
  - Support for 8 external and 34 internal discrete interrupt sources
  - Support for one external (optional) and seven internal machine check interrupt sources
  - Programmable highest priority request
  - Four groups of interrupts with programmable priority
  - External and internal interrupts directed to host processor
  - Redirects interrupts to external $\overline{\text{INTA}}$ signal when in core disable mode
  - Unique vector number for each interrupt source

- Dual I^2C interfaces
  - Two-wire interface
  - Multiple-master support
  - Master or slave I^2C mode support
  - On-chip digital filtering rejects spikes on the bus
  - System initialization data is optionally loaded from I^2C-1 EPROM by boot sequencer embedded hardware
- DMA (Direct memory access) controller
  - Four independent virtual channels
  - Concurrent execution across multiple channels with programmable bandwidth control
  - Handshaking (external control) signals supported for all channels: $\overline{\text{DMA_DREQ}}$[0:3], $\overline{\text{DMA_DACK}}$[0:3], $\overline{\text{DMA_DDONE}}$[0:3]
  - All channels accessible by local core and remote PCI masters
  - Misaligned transfer capability for source/destination address
  - Data chaining and direct mode
  - Interrupt on completed segment and chain
- DUART
  - Two 4-wire interfaces (RxD, TxD, $\overline{\text{RTS}}$, $\overline{\text{CTS}}$)
  - Programming model compatible with the original 16450 UART and the PC16550D
- Serial peripheral interface (SPI)
  - Master or slave support
- Parallel I/O
  - General-purpose I/O (GPIO)
    - 64 parallel I/O pins multiplexed on various chip interfaces
  - Open drain capability
  - Interrupt capability
- System timers
  - Periodic interrupt timer
  - Real-time clock
  - Software watchdog timer
  - Eight general-purpose timers
- IEEE Std. 1149.1™ compliant JTAG boundary scan
- Integrated PCI bus and SDRAM clock generation

1.3 MPC8349EA Architecture Overview

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

1.3.1 PowerPC Core

The MPC8349EA PowerQUICC II Pro processor contains the e300c1 PowerPC processor core, which is an enhanced version of the MPC603e core (used in previous generations of PowerQUICC II processors). Enhancements include twice as much L1 cache (32-Kbyte data cache and 32-Kbyte instruction cache) with integrated parity checking and other performance-enhancing features. The e300c1 core is upward software-compatible with existing MPC603e core-based products.

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

- The *e300 PowerPC™ Core 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 e300c1 core is a low-power implementation of the family of microprocessors that implements the PowerPC architecture. The core implements the 32-bit portion of the PowerPC architecture, which 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 retire as many as five instructions in execution per clock cycle. Instructions can execute out of order for increased performance; however, the core makes completion appear sequential.

The e300c1 core integrates five execution units—an integer unit (IU) 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, 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. On the e300c1 core, the FPU is pipelined so a single-precision multiply-add instruction can be issued and completed every clock cycle.

The e300c1 core provides independent on-chip, 32-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 TLBs and caches 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 PowerPC architecture, if an effective address hits in both the TLB and BAT array, the BAT translation takes priority.

As an added feature to the e300c1 core, the device can lock the contents of one to all 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 e300c1 core has high-performance 64-bit data bus and 32-bit address bus interfaces to the rest of the device. The e300c1 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 e300c1 core that shows how the execution units (IU, FPU, BPU, LSU, and SRU) operate independently and in parallel. Note that this is a conceptual diagram and does not attempt to show how these features are physically implemented on the chip.

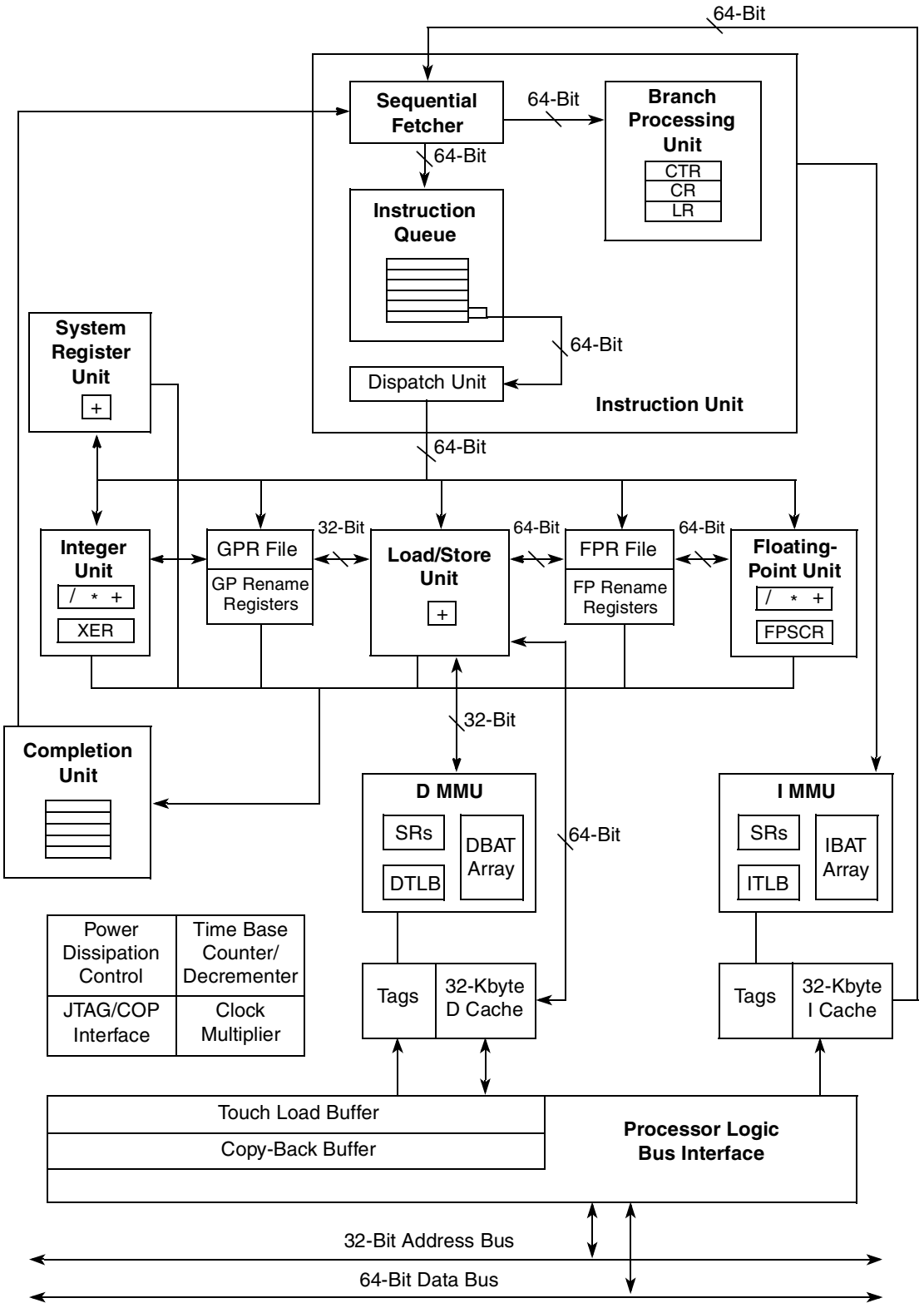


Figure 1-2. MPC8349EA Integrated e300c1 Core Block Diagram

1.3.2 Security Engine

A hardware encryption block is also integrated in the device. It supports many encryption algorithms allowing for high performance data encryption and authentication as required in today's SoHo/RoBo routers. The encryption block is compatible with the corresponding block in the MPC8280.

The security engine supports DES, 3DES, MD-5, SHA-1, AES, PKEU, RNG, and RC-4 encryption algorithms in hardware. It also includes XOR parity generation acceleration for RAID applications.

A block diagram of the security engine's internal architecture is shown in Figure 1-3. The bus interface module is designed to transfer 64-bit words between the internal bus and any register inside the security engine.

An operation begins with a write of a pointer to a crypto-channel fetch register that points to a data packet descriptor. The channel requests the descriptor and decodes the operation to be performed. The channel then requests the controller to assign crypto execution units and fetch the keys, IVs, and data needed to perform the given operation. The controller satisfies the requests by assigning execution units to the channel and by making requests to the master interface. As data is processed, it is written to the individual execution unit's output buffer and then back to system memory through the bus interface module.

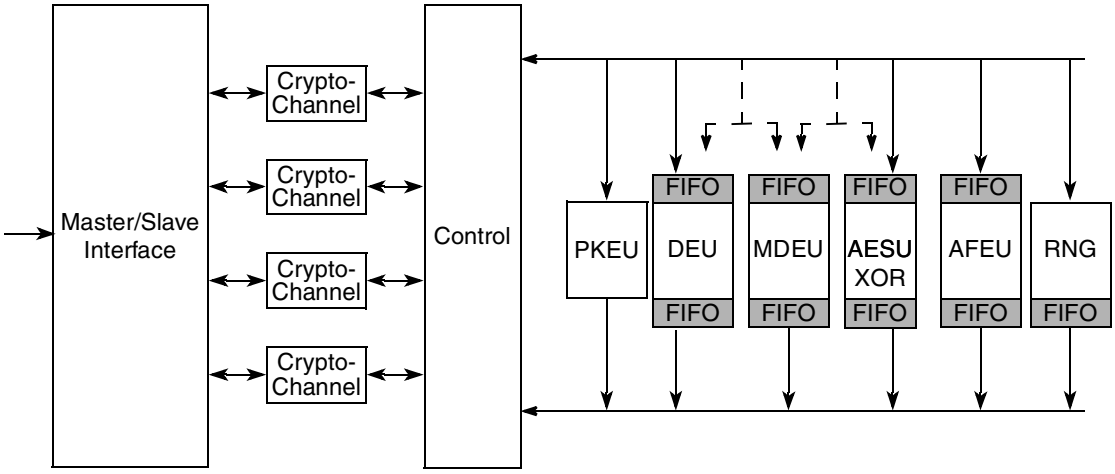


Figure 1-3. Integrated Security Engine Functional Blocks

1.3.3 DDR Memory Controller

This fully programmable DDR SDRAM controller supports most JEDEC standard x8 or x16 DDR1 or DDR2 memories available today, including buffered and unbuffered DIMMs. However, mixing nonregistered and registered DIMMs in the same system is not supported. The built-in error checking and correction (ECC) ensures very low bit-error rates for reliable high-frequency operation. Dynamic power management and auto-precharge modes simplify memory system design. Special features such as ECC error injection support rapid system debug.

The DDR memory controller includes the following features:

- Support for DDR1 and DDR2 SDRAM
- 32-/40-bit and 64/72-bit SDRAM data bus

- Programmable settings for meeting all SDRAM timing parameters
- Many different SDRAM configurations supported
  - Support for as many as four physical banks (chip selects), each bank independently addressable
  - Support for 64-Mbit to 4-Gbit devices with x8 or x16 data ports (no direct x4 support)
  - Support for unbuffered and registered DIMMs
- Support for data mask signals and read-modify-write operations for sub-double word writes
- Support for double-bit error detection and single-bit error correction ECC (8-bit check word across 64- or 32-bit data when in 32-bit mode)
- Two-entry input request queue
- Open page management (dedicated entry for each sub-bank)
- Memory controller clock frequency of two times the SDRAM clock with support for sleep power management mode
- Support for error injection on ECC

1.3.4 Dual Three-Speed Ethernet Controllers

The three-speed Ethernet controllers (TSECs) are designed to support 10/100 Ethernet to 1000 Mbps, or 1 Gbps IEEE Std. 802.3 protocol. The Ethernet IEEE Std. 802.3 protocol is a widely used LAN, based on the carrier-sense multiple access/collision detect (CSMA/CD) approach. Because Ethernet and IEEE Std. 802.3 protocols are similar and can coexist on the same LAN, 10/100 Ethernet provides increased Ethernet speed from 10 to 100 Mbps and provides a simple, cost-effective option for backbone and server connectivity. The TSEC offers gigabit Ethernet, which builds on top of the Ethernet protocol, but increases speed tenfold over 10/100 Ethernet to 1 Gbps.

The TSECs include these distinctive features:

- Ethernet media access controller (MAC)
- First-in first-out (FIFO) controller
- Direct memory access (DMA) controller
- Ten-bit interface (TBI)
- Register-based statistical module that supports management information base (MIB) remote monitoring (RMON)

The most-significant byte of data in a receive or transmit data buffer corresponds to the most-significant byte of a frame, respectively.

The complete TSEC is designed for single MAC applications. The TSECs support several standard MAC-PHY interfaces to connect to an external Ethernet transceiver:

- MII interface running at 10/100 Mbps
- GMII interface running at 1000 Mbps
- TBI interface that can be connected to a SerDes device for fibre channel applications.
- Reduced pin count versions of the GMII (RGMII) and ten-bit (RTBI) interfaces

The TSEC software programming model is similar to the MPC8260 (PowerQUICC II) device. Hence, it enables Freescale customers to leverage already implemented Ethernet drivers, reducing the software development cycle.

1.3.5 PCI Controllers

The 32-/64-bit PCI controllers are compatible with the *PCI Local Bus Specification, Rev. 2.3*. Both PCI1 and PCI2 interfaces can function as host bridge interfaces. The PCI1 interface can optionally function as an agent device. Both PCI controllers support 32-bit addressing and 32-bit data buses. The PCI1 controller also supports a 64-bit data bus.

As a master, the device supports read and write operations to the PCI memory space, the PCI I/O space, and the PCI configuration space. Also, the device can generate PCI special-cycle and interrupt acknowledge commands. As a target, the device supports read and write operations to system memory, as well as PCI configuration space and the on-chip memory mapped configuration space.

The device PCI controllers include the following distinctive features:

- Address stepping on configuration transactions
- Fast back-to-back transactions
- Data streaming
- When in host mode, the PCI controllers support external signal isolation, thus enabling power shut off to external devices

1.3.5.1 PCI Bus Arbitration Unit

The PCI controllers each contain a PCI bus arbitration unit, which eliminates the need for an external unit, thus lowering system complexity and cost. It has the following features:

- PCI1 supports five $\overline{\text{REQ}}$/$\overline{\text{GNT}}$ signal pairs, thus supporting five external masters. The device PCI1 controller is the sixth member of the arbitration pool.
- PCI2 supports three $\overline{\text{REQ}}$/$\overline{\text{GNT}}$ signal pairs, thus supporting three external masters. The PCI1 controller is the fourth member of the arbitration pool.
- The bus arbitration unit allows fairness as well as a priority mechanism.
- A two-level round-robin scheme is used in which each device can be programmed within a pool of a high- or low-priority arbitration. One member of the low-priority pool is promoted to the high-priority pool. As soon as it is granted the bus, it returns to the low-priority pool.
- The unit can be disabled to allow a remote arbitration unit to be used.
- The unit can be isolated to allow power shut off of external devices.

1.3.6 Universal Serial Bus (USB) 2.0

The USB 2.0 controller offers simultaneous operation as host and device. The USB host and device controllers provide point-to-point connectivity, which complies with the USB Specification, Rev. 2.0. It also allows the host and device functions to co-exist and operate simultaneously. The USB controllers can

be configured to operate as stand-alone hosts, stand-alone devices, dual hosts, or both host and device functions that are operating simultaneously. See Figure 1-4 for more information.

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

- Bulk
- Control
- Interrupt
- Isochronous

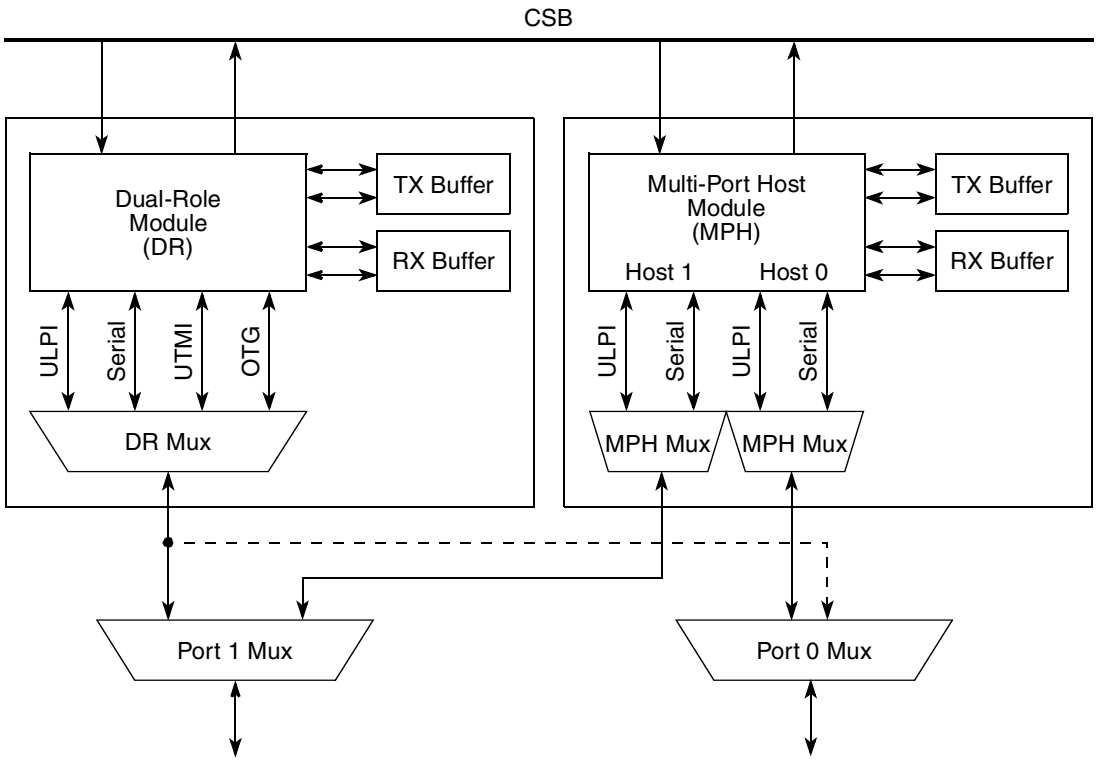


Figure 1-4. USB Controllers Port Configuration

1.3.6.1 USB Dual-Role Controller

- Supports USB on-the-go mode, which includes both device and host functionality
- Complies with USB Specification, Rev. 2.0
- Supports operation as a stand-alone USB device
  - Supports one upstream facing port
  - Supports six programmable bidirectional USB endpoints
- 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 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 UTMI, serial and UTMI + low-pin interface (ULPI)

1.3.6.2 USB Multi-Port Host Controller

- Supports operation as a stand-alone USB host controller
  - Supports USB root hub with one or two downstream-facing ports
  - Enhanced host controller interface (EHCI) compatible
- Complies with USB Specification, Rev. 2.0
- Supports high-speed (480 Mbps), full-speed (12 Mbps), and low-speed (1.5 Mbps) operations
- Supports a direct connection to a high-speed device without an external hub
- Supports external PHY with serial and low-pin count (ULPI) interfaces

1.3.7 Local Bus Controller (LBC)

The main component of the local bus controller (LBC) is its memory controller, which provides a seamless interface to many types of memory devices and peripherals. The memory controller is responsible for controlling eight memory banks shared by a high performance SDRAM machine, 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 synchronous DRAM (SDRAM), SRAM, EPROM, Flash EPROM, burstable RAM, and other peripherals. The LBC external address latch enable (LALE) signal allows multiplexing of addresses with data signals to reduce the device pin count.

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

- Memory controller with eight memory banks (chip selects)
  - 32-bit address decoding with mask
  - Variable memory block sizes (32 Kbytes to 2 Gbytes)
  - Selection of control signal generation on a per-bank basis
  - Data buffer controls activated on a per-bank basis
  - Odd/even parity checking including read-modify-write (RMW) parity for single accesses
  - Parity byte-select
- Synchronous DRAM (SDRAM) machine
  - Provides the control functions and signals for glueless connection to JEDEC-compliant SDRAM devices
  - Supports up to four concurrent open pages
  - Supports SDRAM port size of 32-, 16-, and 8-bit
  - Supports external address and/or command line buffering

- General-purpose chip-select machine (GPCM)
  - Compatible with SRAM, EPROM, FEPROM, and peripherals
  - Global (boot) chip-select available at system reset
  - Boot chip-select support for 8-, 16-, 32-bit devices
  - Minimum three-clock access to external devices
  - Four byte-write-enable signals ($\overline{\text{LWE}}$[0:3])
- 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
  - User-specified control-signal patterns can be initiated by software
  - Support for 8-, 16-, 32-bit devices
  - Page mode support for successive transfers within a burst
- Delay locked loop (DLL) with software-configurable bypass for generating low-frequency bus clocks

1.3.8 Integrated Programmable Interrupt Controller (IPIC)

The 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 8 external and 35 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
- Two 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
- Ability to redirect interrupts to external $\overline{\text{INTA}}$ pin when in core disable mode

1.3.9 Dual I^{2}C Interfaces

The inter-IC (IIC or I^{2}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^2C allows the connection of additional devices to the bus for expansion and system development.

The I^2C 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^2C controller consists of a transmitter/receiver unit, clocking unit, and control unit. The I^2C unit supports general broadcast mode and on-chip filtering rejects spikes on the bus.

The I^2C interfaces include 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
- System initialization data is optionally loaded from I^2C EPROM by boot sequencer embedded hardware

1.3.10 DMA Controller

The DMA engine is capable of transferring blocks of data from any legal address range to any other legal address range. Therefore, it can perform a DMA transfer between any of its I/O or memory ports, or even between two devices or locations on the same port.

The DMA controller offers the following features:

- Four high-speed/high-bandwidth channels accessible by local and remote masters
- Handshaking (external control) signals supported for all channels: $\overline{\text{DREQ}}$[0:3], $\overline{\text{DACK}}$[0:3], $\overline{\text{DDONE}}$[0:3]
- Basic DMA operation modes (direct, simple chaining)
- Support for misaligned transfers
- Programmable bandwidth control between channels
- Interrupt on error and completed segment or chain

1.3.11 Dual Universal Asynchronous Receiver/Transmitter (DUART)

The MPC8349EA includes a DUART intended for use in maintenance, bring up, and debug systems. The device provides a standard four-wire handshake (TXD, RXD, $\overline{\text{RTS}}$, $\overline{\text{CTS}}$) for each port. The DUART is a slave interface. An interrupt is provided to the interrupt controller or optionally steered externally to allow device handshakes. 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
- Clear to send ($\overline{\text{CTS}}$) and ready to send ($\overline{\text{RTS}}$) MODEM control functions
- 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

1.3.12 Serial Peripheral Interface (SPI)

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

The 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.

1.3.13 System Timers

The system includes the following timers:

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

1.4 Applications

The MPC8349EA can be used for control processing in applications such as wireless LAN, routers/switches, line cards, embedded computing, multi-channel modems, network storage, image display systems, enterprise I/O processor, internet access device (IAD), disk controller for RAID systems, and copier/printer board control.

1.5 Differences Between MPC8349E and MPC8349EA

Table 1-2 compares features of the MPC8349E to those of the MPC8349EA.

Table 1-2. Differences Between MPC8349E and MPC8349EA

Description	MPC8349E	MPC8349EA
DMA		
External handshake signal support	No	Yes
DDR		
Supports DDR2	No	Yes
Battery backup support	No	Yes
Local Bus		
Number of chip selects (banks) supported	4	8
PCI		
PCI version supported	Rev 2.2	Rev 2.3
Security		
NIST SHA-224 support	No	Yes
Raid XOR functionality supported	No	Yes

Chapter 2
Memory Map

This chapter describes the MPC8349EA 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.

2.1 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 relocatable 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 5.2.4.1, “Internal Memory Map Registers Base Address Register (IMMRBAR)”, for more information. The default value for IMMRBAR is 0xFF40_0000.

2.2 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.

2.3 Complete IMMR Map

Table 2-1 lists the memory-mapped register regions (windows).

Reading from address locations which appear as reserved in the memory map table is not guaranteed to return predictable data. Writing to address locations which 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 will be 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.

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 will indicate when this is needed.

Table 2-1. IMMR Memory Map

Address	Use	Actual Size	Window	Cross Reference
0x0_0000–0x0_01FF	System configuration	512 bytes	512 bytes	Table 2-2
0x0_0200–0x0_02FF	Watchdog timer	16 bytes	256 bytes	Table 2-2
0x0_0300–0x0_03FF	Real time clock	32 bytes	256 bytes	Table 2-2
0x0_0400–0x0_04FF	Periodic interval timer	32 bytes	256 bytes	Table 2-2
0x0_0500–0x0_05FF	Global timers module 1	64 bytes	256 bytes	Table 2-2
0x0_0600–0x0_06FF	Global timers module 2	64 bytes	256 bytes	Table 2-2
0x0_0700–0x0_07FF	Integrated programable interrupt controller (IPIC)	128 bytes	256 bytes	Table 2-2
0x0_0800–0x0_08FF	System arbiter	30 bytes	256 bytes	Table 2-2
0x0_0900–0x0_09FF	Reset module		256 bytes	Table 2-2
0x0_0A00–0x0_0AFF	Clock module	44 bytes	256 bytes	Table 2-2
0x0_0B00–0x0_0BFF	Power management control module		256 bytes	Table 2-2
0x0_0C00–0x0_0CFF	GPIO 1	24 bytes	256 bytes	Table 2-2
0x0_0D00–0x0_0DFF	GPIO 2	24 bytes	256 bytes	Table 2-2
0x0_0E00–0x0_0EFF	Reserved, should be cleared	—	256 bytes	
0x0_0F00–0x0_0FFF	Reserved, should be cleared	20 bytes	256 bytes	
0x0_1000–0x0_10FF	Clock control DDR	20 bytes	256 bytes	Table 2-2
0x0_1100–0x0_11FF	DLL LBC	20 bytes	256 bytes	Table 2-2
0x0_1200–0x0_12FF	Reserved, should be cleared	—	256 bytes	
0x0_1300– 0x0_1FFF	Reserved, should be cleared	—	1280 bytes	
0x0_2000–0x0_2FFF	DDR MEMC	3.8 Kbytes	4 Kbytes	Table 2-2
0x0_3000–0x0_30FF	I^2C1 controller	24 bytes	256 bytes	Table 2-2
0x0_3100–0x0_31FF	I^2C2 controller	24 bytes	256 bytes	Table 2-2
0x0_3200–0x_03FFF	Reserved, should be cleared	—	3.5 Kbytes	
0x0_4000–0x0_44FF	Reserved, should be cleared	—	—	
0x0_4500–0x0_46FF	DUART	18 bytes x 2	4 Kbytes	Table 2-2
0x0_4700–0x0_4FFF	Reserved, should be cleared	—	—	
0x0_5000–0x0_5FFF	LBC	224 bytes	4 Kbytes	Table 2-2
0x0_6000–0x0_6FFF	Reserved, should be cleared	—	4 Kbytes	
0x0_7000–0x0_7FFF	SPI	24 bytes	4 Kbytes	Table 2-2
0x0_8000–0x0_82FF	DMA	768 bytes	768 bytes	Table 2-2
0x0_8300–0x0_837F	PCI1 configuration	16 bytes	128 bytes	Table 2-2
0x0_8380–0x0_83FF	PCI2 configuration	16 bytes	128 bytes	Table 2-2
0x0_8400–0x0_84FF	IOS	256 bytes	256 bytes	Table 2-2

Table 2-1. IMMR Memory Map (continued)

Address	Use	Actual Size	Window	Cross Reference
0x0_8500–0x0_85FF	PCI1 controller	128 bytes	256 bytes	Table 2-2
0x0_8600–0x0_86FF	PCI2 controller	128 bytes	256 bytes	Table 2-2
0x0_8700–0x2_1FFF	Reserved, should be cleared	—	—	
0x2_2000–0x2_2FFF	USB MPH	1 Kbyte	4 Kbytes	Table 2-2
0x2_3000–0x2_3FFF	USB_DR module	1 Kbyte	4 Kbytes	Table 2-2
0x2_4000–0x2_4FFF	TSEC 1	3 Kbytes + 1K reserved	4 Kbytes	Table 2-2
0x2_5000–0x2_5FFF	TSEC 2	3 Kbytes + 1K reserved	4 Kbytes	Table 2-2
0x2_6000–0x2_FFFF	Reserved, should be cleared	—	—	
0x3_0000–0x3_FFFF	Security engine	52 Kbytes	64 Kbytes	Table 2-2
0x4_0000–0xF_FFFF	Reserved, should be cleared	—	—	

Table 2-2 lists the memory-mapped registers.

Table 2-2. Memory Map

Offset	Register	Access	Reset	Section/Page
System Configuration Registers				
0x0_0000	IMMRBAR—Internal memory map base address register	R/W	0xFF40_0000	5.2.4.1/5-5
0x0_0004	Reserved, should be cleared	—	—	—
0x0_0008	ALTCBAR—Alternate configuration base address register	R/W	0x0000_0000	5.2.4.2/5-7
0x0_000C–0x0_001C	Reserved, should be cleared	—	—	—
0x0_0020	LBLAWBAR0—LBC local access window 0 base address register	R/W	0x0000_0000[1]	5.2.4.3/5-7
0x0_0024	LBLAWAR0—LBC local access window 0 attribute register	R/W	0x0000_0000[2]	5.2.4.4/5-8
0x0_0028	LBLAWBAR1—LBC local access window 1 base address register	R/W	0x0000_0000	5.2.4.3/5-7
0x0_002C	LBLAWAR1—LBC local access window 1 attribute register	R/W	0x0000_0000	5.2.4.4/5-8
0x0_0030	LBLAWBAR2—LBC local access window 2 base address register	R/W	0x0000_0000	5.2.4.3/5-7
0x0_0034	LBLAWAR2—LBC local access window 2 attribute register	R/W	0x0000_0000	5.2.4.4/5-8
0x0_0038	LBLAWBAR3—LBC local access window 3 base address register	R/W	0x0000_0000	5.2.4.3/5-7
0x0_003C	LBLAWAR3—LBC local access window 3 attribute register	R/W	0x0000_0000	5.2.4.4/5-8
0x0_0040–0x0_005C	Reserved, should be cleared	—	—	—
0x0_0060	PCILAWBAR0—PCI local access window0 base address register	R/W	0x0000_0000[3]	5.2.4.5/5-9

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x0_0064	PCILAWAR0—PCI local access window0 attribute register	R/W	0x0000_0000[4]	5.2.4.6/5-10
0x0_0068	PCILAWBAR1—PCI local access window1 base address register	R/W	0x0000_0000	5.2.4.5/5-9
0x0_006C	PCILAWAR1—PCI local access window1 attribute register	R/W	0x0000_0000	5.2.4.6/5-10
0x0_0070–0x0_009C	Reserved, should be cleared	—	—	—
0x0_00A0	DDRLAWBAR0—DDR local access window0 base address register	R/W	0x0000_0000[5]	5.2.4.7/5-12
0x0_00A4	DDRLAWAR0—DDR local access window0 attribute register	R/W	0x0000_0000[6]	5.2.4.8/5-13
0x0_00A8	DDRLAWBAR1—DDR local access window1 base address register	R/W	0x0000_0000	5.2.4.7/5-12
0x0_00AC	DDRLAWAR1—DDR local access window1 attribute register	R/W	0x0000_0000	5.2.4.8/5-13
0x0_00B0–0x0_00FC	Reserved, should be cleared	—	—	—
0x0_0100	System general purpose register low (SGPRL)	R/W	0x0000_0000	5.3.2.1/5-17
0x0_0104	System general purpose register high (SGPRH)	R/W	0x0000_0000	5.3.2.2/5-17
0x0_0108	System part and revision ID register (SPRIDR)	R	0x0000_0000	5.3.2.3/5-18
0x0_010C	Reserved, should be cleared	—	—	—
0x0_0110	System priority configuration register (SPCR)	R/W	0x0000_0000	5.3.2.4/5-19
0x0_0114	System I/O configuration register low (SICRL)	R/W	0x0000_0000	5.3.2.5/5-21
0x0_0118	System I/O configuration register high (SICRH)	R/W	0x0000_0000[7]	5.3.2.6/5-24
0x0_011C–0x0_0124	Reserved	—	—	—
0x0_0128	DDR control driver register (DDRCDR)	R/W	0x0004_0000	5.3.2.8/5-28
0x0_012C	DDR debug status register (DDRDSR)	R	0x3300_0000	5.3.2.9/5-30
0x0_0130–0x0_01FC	Reserved	—	—	—
Watchdog Timer (WDT) Registers				
0x0_0200–0x0_0203	Reserved, should be cleared	—	—	—
0x0_0204	SWCRR—System watchdog control register	R/W	0x0000_0007	5.4.4.1/5-33
0x0_0208	SWCNR—System watchdog count register	R	0x0000_FFFF	5.4.4.2/5-34
0x0_020C–0x0_020D	Reserved, should be cleared	—	—	—
0x0_020E	SWSRR—System watchdog service register	R/W	0x0000_0000	5.4.4.3/5-34
Real Time Clock Module Registers (RTC)				
0x0_0300	RTCNR—Real time counter control register	R/W	0x0000_0000	5.5.5.1/5-40
0x0_0304	RTLDR—Real time counter load register	R/W	0x0000_0000	5.5.5.2/5-41

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x0_0308	RTPSR—Real time counter prescale register	R/W	0x0000_0000	5.5.5.3/5-41
0x0_030C	RTCTR—Real time counter register	R	0x0000_0000	5.5.5.4/5-42
0x0_0310	RTEVR—Real time counter event register	R/W	0x0000_0000	5.5.5.5/5-42
0x0_0314	RTALR—Real time counter alarm register	R/W	0xFFFF_FFFF	5.5.5.6/5-43
0x0_0314–0x0_031F	Reserved, should be cleared	—	—	—
Periodic Interval Timer (PIT) Registers				
0x0_0400	PTCNR—Periodic interval timer control register	R/W	0x0000_0000	5.6.5.1/5-47
0x0_0404	PTLDR—Periodic interval timer load register	R/W	0x0000_0000	5.6.5.2/5-48
0x0_0408	PTPSR—Periodic interval timer prescale register	R/W	0x0000_0000	5.6.5.3/5-49
0x0_040C	PTCTR—Periodic interval timer counter register	R	0x0000_0000	5.6.5.4/5-49
0x0_0410	PTEVR—Periodic interval timer event register	R/W	0x0000_0000	5.6.5.5/5-50
0x0_0410–0x0_041F	Reserved, should be cleared	—	—	—
Global Timers Module 1				
0x0_0500	GTCFR1—Timer 1 and 2 global timers configuration register	R/W	0x00	5.7.5.1/5-57
0x0_0501–0x0_0503	Reserved, should be cleared	—	—	—
0x0_0504	GTCFR2—Timer 3 and 4 global timers configuration register	R/W	0x00	5.7.5.1/5-57
0x0_0505–0x0_050F	Reserved, should be cleared	—	—	—
0x0_0510	GTMDR1—Timer 1 global timers mode register	R/W	0x0000	5.7.5.2/5-60
0x0_0512	GTMDR2—Timer 2 global timers mode register			
0x0_0514	GTRFR1—Timer 1 global timers reference register	R/W	0x0000	5.7.5.3/5-62
0x0_0516	GTRFR2—Timer 2 global timers reference register			
0x0_0518	GTCPR1—Timer 1 global timers capture register	R/W	0x0000	5.7.5.4/5-62
0x0_051A	GTCPR2—Timer 2 global timers capture register			
0x0_051C	GTCNR1—Timer 1 global timers counter register	R/W	0x0000	5.7.5.5/5-63
0x0_051E	GTCNR2—Timer 2 global timers counter register			
0x0_0520	GTMDR3—Timer 3 global timers mode register	R/W	0x0000	5.7.5.2/5-60
0x0_0522	GTMDR4—Timer 4 global timers mode register			
0x0_0524	GTRFR3—Timer 3 global timers reference register	R/W	0x0000	5.7.5.3/5-62
0x0_0526	GTRFR4—Timer 4 global timers reference register			
0x0_0528	GTCPR3—Timer 3 global timers capture register	R	0x0000	5.7.5.4/5-62
0x0_052A	GTCPR4—Timer 4 global timers capture register			

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x0_052C	GTCNR3—Timer 3 global timers counter register	R/W	0x0000	5.7.5.5/5-63
0x0_052E	GTCNR4—Timer 4 global timers counter register			
0x0_0530	GTEVR1—Timer 1 global timers event register	Special	0x0000	5.7.5.6/5-63
0x0_0532	GTEVR2—Timer 2 global timers event register			
0x0_0534	GTEVR3—Timer 3 global timers event register			
0x0_0536	GTEVR4—Timer 4 global timers event register			
0x0_0538	GTPSR1—Timer 1 global timers prescale register	R/W	0x0003	5.7.5.7/5-64
0x0_053A	GTPSR2—Timer 2 global timers prescale register			
0x0_053C	GTPSR3—Timer 3 global timers prescale register			
0x0_053E	GTPSR4—Timer 4 global timers prescale register			
Global Timers Module 2				
0x0_0600	GTCFR1—Timer 1 and 2 global timers configuration register	R/W	0x00	5.7.5.1/5-57
0x0_0601–0x0_0603	Reserved, should be cleared	—	—	—
0x0_0604	GTCFR2—Timer 3 and 4 global timers configuration register	R/W	0x00	5.7.5.1/5-57
0x0_0605–0x0_060F	Reserved, should be cleared	—	—	—
0x0_0610	GTMDR1—Timer 1 global timers mode register	R/W	0x0000	5.7.5.2/5-60
0x0_0612	GTMDR2—Timer 2 global timers mode register			
0x0_0614	GTRFR1—Timer 1 global timers reference register	R/W	0x0000	5.7.5.3/5-62
0x0_0616	GTRFR2—Timer 2 global timers reference register			
0x0_0618	GTCPR1—Timer 1 global timers capture register	R/W	0x0000	5.7.5.4/5-62
0x0_061A	GTCPR2—Timer 2 global timers capture register			
0x0_061C	GTCNR1—Timer 1 global timers counter register	R/W	0x0000	5.7.5.5/5-63
0x0_061E	GTCNR2—Timer 2 global timers counter register			
0x0_0620	GTMDR3—Timer 3 global timers mode register	R/W	0x0000	5.7.5.2/5-60
0x0_0622	GTMDR4—Timer 4 global timers mode register			
0x0_0624	GTRFR3—Timer 3 global timers reference register	R/W	0x0000	5.7.5.3/5-62
0x0_0626	GTRFR4—Timer 4 global timers reference register			
0x0_0628	GTCPR3—Timer 3 global timers capture register	R	0x0000	5.7.5.4/5-62
0x0_062A	GTCPR4—Timer 4 global timers capture register			
0x0_062C	GTCNR3—Timer 3 global timers counter register	R/W	0x0000	5.7.5.5/5-63
0x0_062E	GTCNR4—Timer 4 global timers counter register			

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x0_0630	GTEVR1—Timer 1 global timers event register	Special	0x0000	5.7.5.6/5-63
0x0_0632	GTEVR2—Timer 2 global timers event register			
0x0_0634	GTEVR3—Timer 3 global timers event register			
0x0_0636	GTEVR4—Timer 4 global timers event register			
0x0_0638	GTPSR1—Timer 1 global timers prescale register	R/W	0x0003	5.7.5.7/5-64
0x0_063A	GTPSR2—Timer 2 global timers prescale register			
0x0_063C	GTPSR3—Timer 3 global timers prescale register			
0x0_063E	GTPSR4—Timer 4 global timers prescale register			
Integrated Programmable Interrupt Controller (IPIC)				
0x0_0700	SICFR—System global interrupt configuration register	R/W	0x0000_0000	8.5.1/8-8
0x0_0704	SIVCR—System global interrupt vector register	R	0x0000_0000	8.5.2/8-9
0x0_0708	SIPNR_H—System internal interrupt pending register	R	0x0000_0000	8.5.3/8-11
0x0_070C	SIPNR_L—System internal interrupt pending register	R	0x0000_0000	8.5.3/8-11
0x0_0710	SIPRR_A—System internal interrupt group A priority register	R/W	0x0530_9770	8.5.4/8-14
0x0_0714	Reserved, should be cleared	—	—	—
0x0_0718	Reserved, should be cleared	—	—	—
0x0_071C	SIPRR_D—System internal interrupt group D priority register	R/W	0x0530_9770	8.5.5/8-14
0x0_0720	SIMSR_H—System internal interrupt mask register	R/W	0x0000_0000	8.5.6/8-15
0x0_0724	SIMSR_L—System internal interrupt mask register	R/W	0x0000_0000	8.5.6/8-15
0x0_072C	SEPNR—System external interrupt pending register	R/W	Special	8.5.8/8-18
0x0_0730	SMPRR_A—System mixed interrupt group A priority register	R/W	0x0530_9770	8.5.9/8-18
0x0_0734	SMPRR_B—System mixed interrupt group B priority register	R/W	0x0530_9770	8.5.10/8-19
0x0_0738	SEMSR—System external interrupt mask register	R/W	0x0000_0000	8.5.11/8-20
0x0_073C	SECNR—System external interrupt control register	R/W	0x0000_0000	8.5.12/8-21
0x0_0740	SERSR—System error status register	R/W	0x0000_0000	8.5.13/8-22
0x0_0744	SERMR—System error mask register	R/W	—	8.5.14/8-23
0x0_0748	SERCR—System error control register	R/W	0x0000_0000	8.5.15/8-24
0x0_074C–0x0_074F	Reserved, should be cleared	—	—	—
0x0_0750	SIFCR_H—System internal interrupt force register	R/W	0x0000_0000	8.5.16/8-25
0x0_0754	SIFCR_L—System internal interrupt force register	R/W	0x0000_0000	8.5.16/8-25
0x0_0758	SEFCR—System external interrupt force register	R/W	0x0000_0000	8.5.17/8-26
0x0_075C	SERFR—System error force register	R/W	0x0000_0000	8.5.18/8-26
0x0_0760	SCVCR—System critical interrupt vector register	R	0x0000_0000	8.5.19/8-27
0x0_0764	SMVCR—System management interrupt vector register	R	0x0000_0000	8.5.20/8-27

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x0_0760–0x0_07FF	Reserved, should be cleared	—	—	—
System Arbiter Registers				
0x0_0800	ACR—Arbiter configuration register	R/W	0x0000_0000	6.2.1/6-2
0x0_0804	ATR—Arbiter timers register	R/W	0x00FF_00FF	6.2.2/6-4
0x0_0808	Reserved, should be cleared	R	0x0000_0000	—
0x0_080C	AER—Arbiter event register	R/W	0x0000_0000	6.2.3/6-5
0x0_0810	AIDR—Arbiter interrupt definition register	R/W	0x0000_0000	6.2.4/6-6
0x0_0814	AMR—Arbiter mask register	R/W	0x0000_0000	6.2.5/6-7
0x0_0818	AEATR—Arbiter event attributes register	R	0x0000_0000	6.2.6/6-7
0x0_081C	AEADR—Arbiter event address register	R	0x0000_0000	6.2.7/6-9
0x0_0820	AERR—Arbiter event response register	R/W	0x0000_0000	6.2.8/6-10
Reset Module				
0x0_0900	RCWLR—Reset configuration word low register	R	0x0000_0000	4.5.1.1/4-32
0x0_0904	RCWHR—Reset configuration word high register	R	0x0000_0000	4.5.1.2/4-32
0x0_0908–0x0_090C	Reserved, should be cleared	—	—	—
0x0_0910	RSR—Reset status register	R/W	0x0000_0000	4.5.1.3/4-33
0x0_0914	RMR—Reset mode register	R/W	0x0000_0000	4.5.1.4/4-34
0x0_0918	RPR—Reset protection register	R/W	0x0000_0000	4.5.1.5/4-35
0x0_091C	RCR—Reset control register	R/W	0x0000_0000	4.5.1.6/4-36
0x0_0920	RCER—Reset control enable register	R/W	0x0000_0000	4.5.1.7/4-36
0x0_0924–0x0_09FC	Reserved, should be cleared	—	—	—
Clock Module				
0x0_0A00	SPMR—System PLL mode register	R	0x0000_0000	4.5.2.1/4-37
0x0_0A04	OCCR—Output clock control register	R/W	0x0000_0000	4.5.2.2/4-38
0x0_0A08	SCCR—System clock control register	R/W	0xFFFF_FFFF	4.5.2.3/4-40
0x0_0A08–0x0_0AFC	Reserved, should be cleared	—	—	—
Power Management Control Module				
0x0_0B00	PMCCR—Power management controller configuration register	R/W	0x0000_0000	5.8.3.1/5-69
0x0_0B04	PMCER—Power management controller event register	R/W	0x0000_0000	5.8.3.2/5-70
0x0_0B08	PMCMR—Power management controller mask register	R/W	0x0000_0000	5.8.3.3/5-71
0x0_0B10–0x0_0BFC	Reserved, should be cleared	—	—	—

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
GPIO1 Registers				
0x0_0C00	GP1DIR—GPIO1 direction register	R/W	0x0000_0000	21.3.1/21-3
0x0_0C04	GP1DR—GPIO1 open drain register	R/W	0x0000_0000	21.3.2/21-4
0x0_0C08	GP1DAT—GPIO1 data register	R/W	0x0000_0000	21.3.3/21-4
0x0_0C0C	GP1IER—GPIO1 interrupt event register	R/W	Undefined	21.3.4/21-5
0x0_0C10	GP1IMR—GPIO1 interrupt mask register	R/W	0x0000_0000	21.3.5/21-5
0x0_0C14	GP1ICR—GPIO1 external interrupt control register	R/W	0x0000_0000	21.3.6/21-6
0x0_0C1C–0x0_0CFF	Reserved, should be cleared	—	—	—
GPIO2 Registers				
0x0_0D00	GP2DIR—GPIO2 direction register	R/W	0x0000_0000	21.3.1/21-3
0x0_0D04	GP2ODR—GPIO2 open drain register	R/W	0x0000_0000	21.3.2/21-4
0x0_0D08	GP2DAT—GPIO2 data register	R/W	0x0000_0000	21.3.3/21-4
0x0_0D0C	GP2IER—GPIO2 interrupt event register	R/W	Undefined	21.3.4/21-5
0x0_0D10	GP2IMR—GPIO2 interrupt mask register	R/W	0x0000_0000	21.3.5/21-5
0x0_0D14	GP2ICR—GPIO2 external interrupt control register	R/W	0x0000_0000	21.3.6/21-6
0x0_0D1C–0x0_0EFF	Reserved, should be cleared	—	—	—
DLL				
0x0_0F00–0x0_0FFF	Reserved, should be cleared	—	—	—
0x0_1000–0x0_100F	Reserved, should be cleared.	—	—	—
0x0_1010	MCK enable register (MCKENR)	R/W	0xFC00_0000	4.5.3/4-41
0x0_1014–0x0_10FF	Reserved, should be cleared.	—	—	—
0x0_1100	Reserved. Reset value should be preserved.	R/W	0x0500_0280	—
0x0_1104	Reserved. Reset value should be preserved.	R/W	0x8004_0810	—
0x0_1108	DLL override register (DLLOVR)	R/W	0x0000_0000	22.4.1/22-4
0x0_110C	DLL status register (DLLSR)	R	0x0000_0000	22.4.2/22-4
0x0_1110	DLL clock register (DLLCK)	R/W	0xFC00_0000	22.4.3/22-5
0x0_1110–0x0_12FF	Reserved, should be cleared	—	—	—
0x0_1200–0x0_12FF	Reserved, should be cleared	—	—	—
0x0_1300–0x0_1FFF	Reserved, should be cleared	—	—	—

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
DDR Memory Controller Memory Map				
0x0_2000	CS0_BNDS—Chip select 0 memory bounds	R/W	0x0000_0000	9.4.1.1/9-10
0x0_2008	CS1_BNDS—Chip select 1 memory bounds	R/W	0x0000_0000	
0x0_2010	CS2_BNDS—Chip select 2 memory bounds	R/W	0x0000_0000	
0x0_2018	CS3_BNDS—Chip select 3 memory bounds	R/W	0x0000_0000	
0x0_2020–0x0_207F	Reserved, should be cleared	—	—	—
0x0_2080	CS0_CONFIG—Chip select 0 configuration	R/W	0x0000_0000	9.4.1.2/9-11
0x0_2084	CS1_CONFIG—Chip select 1 configuration	R/W	0x0000_0000	
0x0_2088	CS2_CONFIG—Chip select 2 configuration	R/W	0x0000_0000	
0x0_208C	CS3_CONFIG—Chip select 3 configuration	R/W	0x0000_0000	
0x0_2090–0x0_20FF	Reserved	—	—	
0x0_2100	TIMING_CFG_3—DDR SDRAM timing configuration 3	R/W	0x0000_0000	9.4.1.3/9-13
0x0_2104	TIMING_CFG_0—DDR SDRAM timing configuration 0	R/W	0x0011_0105	9.4.1.4/9-14
0x0_2108	TIMING_CFG_1—DDR SDRAM timing configuration 1	R/W	0x0000_0000	9.4.1.5/9-16
0x0_210C	TIMING_CFG_2—DDR SDRAM timing configuration 2	R/W	0x0000_0000	9.4.1.6/9-18
0x0_2110	DDR_SDRAM_CFG—DDR SDRAM control configuration	R/W	0x0200_0000	9.4.1.7/9-20
0x0_2114	DDR_SDRAM_CFG_2—DDR SDRAM control configuration 2	R/W	0x0000_0000	9.4.1.8/9-22
0x0_2118	DDR_SDRAM_MODE—DDR SDRAM mode configuration	R/W	0x0000_0000	9.4.1.9/9-24
0x0_211C	DDR_SDRAM_MODE_2—DDR SDRAM mode configuration 2	R/W	0x0000_0000	9.4.1.10/9-24
0x0_2120	DDR_SDRAM_MD_CNTL—DDR SDRAM mode control	R/W	0x0000_0000	9.4.1.11/9-25
0x0_2124	DDR_SDRAM_INTERVAL—DDR SDRAM interval configuration	R/W	0x0000_0000	9.4.1.12/9-27
0x0_2128	DDR_DATA_INIT—DDR SDRAM data initialization	R/W	0x0000_0000	9.4.1.13/9-28
0x0_2130	DDR_SDRAM_CLK_CNTL—DDR SDRAM clock control	R/W	0x0200_0000	9.4.1.14/9-28
0x0_2140	Reserved	—	—	—
0x0_2148	DDR_INIT_ADDRESS—DDR training initialization address	R/W	0x0000_0000	9.4.1.15/9-29
0x0_2BF8	DDR_IP_REV1—DDR IP block revision 1	R	0x0002_0200	9.4.1.16/9-30
0x0_2BFC	DDR_IP_REV2—DDR IP block revision 2	R	0x0000_0000	9.4.1.17/9-30
0x0_2E00	DATA_ERR_INJECT_HI—Memory data path error injection mask high	R/W	0x0000_0000	9.4.1.18/9-31
0x0_2E04	DATA_ERR_INJECT_LO—Memory data path error injection mask low	R/W	0x0000_0000	9.4.1.19/9-31
0x0_2E08	ECC_ERR_INJECT—Memory data path error injection mask ECC	R/W	0x0000_0000	9.4.1.20/9-32
0x0_2E20	CAPTURE_DATA_HI—Memory data path read capture high	R/W	0x0000_0000	9.4.1.21/9-32

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x0_2E24	CAPTURE_DATA_LO—Memory data path read capture low	R/W	0x0000_0000	9.4.1.22/9-33
0x0_2E28	CAPTURE_ECC—Memory data path read capture ECC	R/W	0x0000_0000	9.4.1.23/9-33
0x0_2E40	ERR_DETECT—Memory error detect	w1c	0x0000_0000	9.4.1.24/9-33
0x0_2E44	ERR_DISABLE—Memory error disable	R/W	0x0000_0000	9.4.1.25/9-34
0x0_2E48	ERR_INT_EN—Memory error interrupt enable	R/W	0x0000_0000	9.4.1.26/9-35
0x0_2E4C	CAPTURE_ATTRIBUTES—Memory error attributes capture	R/W	0x0000_0000	9.4.1.27/9-36
0x0_2E50	CAPTURE_ADDRESS—Memory error address capture	R/W	0x0000_0000	9.4.1.28/9-37
0x0_2E54	Reserved	—	—	—
0x0_2E58	ERR_SBE—Single-Bit ECC memory error management	R/W	0x0000_0000	9.4.1.29/9-37
I^2C1 Controller				
0x0_3000	I2C1ADR—I^2C1 address register	R/W	0x00	17.3.1.1/17-5
0x0_3004	I2C1FDR—I^2C1 frequency divider register	R/W	0x00	17.3.1.2/17-5
0x0_3008	I2C1CR—I^2C1 control register	R/W	0x00	17.3.1.3/17-6
0x0_300C	I2C1SR—I^2C1 status register	R/W	0x81	17.3.1.4/17-8
0x0_3010	I2C1DR—I^2C1 data register	R/W	0x00	17.3.1.5/17-9
0x0_3014	I2C1DFSRR—I^2C1 digital filter sampling rate register	R/W	0x0001_0000	17.3.1.6/17-10
0x0_301C–0x0_30FF	Reserved, should be cleared	—	—	—
I^2C2 Controller				
0x0_3100	I2C2ADR—I^2C2 address register	R/W	0x00	17.3.1.1/17-5
0x0_3104	I2C2FDR—I^2C2 frequency divider register	R/W	0x00	17.3.1.2/17-5
0x0_3108	I2C2CR—I^2C2 control register	R/W	0x00	17.3.1.3/17-6
0x0_310C	I2C2SR—I^2C2 status register	R/W	0x81	17.3.1.4/17-8
0x0_3110	I2C2DR—I^2C2 data register	R/W	0x00	17.3.1.5/17-9
0x0_3114	I2C2DFSRR—I^2C2 digital filter sampling rate register	R/W	0001_0000	17.3.1.6/17-10
0x0_311C–0x0_31FF	Reserved, should be cleared	—	—	—
0x0_3200–0x0_3FFF	Reserved, should be cleared	—	—	—
DUART				
0x0_4000–0x0_44FF	Reserved, should be cleared	—	—	—
0x0_4500	URBR—ULCR[DLAB] = 0 UART1 receiver buffer register	R	0x00	18.3.1.1/18-6
0x0_4500	UTHR—ULCR[DLAB] = 0 UART1 transmitter holding register	W	0x00	18.3.1.2/18-6
0x0_4500	UDLB—ULCR[DLAB] = 1 UART1 divisor least significant byte register	R/W	0x00	18.3.1.3/18-7

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x0_4501	UIER—ULCR[DLAB] = 0 UART1 interrupt enable register	R/W	0x00	18.3.1.4/18-8
0x0_4501	UDMB—ULCR[DLAB] = 1 UART1 divisor most significant byte register	R/W	0x00	18.3.1.3/18-7
0x0_4502	UIIR—ULCR[DLAB] = 0 UART1 interrupt ID register	R	0x01	18.3.1.5/18-9
0x0_4502	UFCR—ULCR[DLAB] = 0 UART1 FIFO control register	W	0x00	18.3.1.6/18-10
0x0_4502	UAFR—ULCR[DLAB] = 1 UART1 alternate function register	R/W	0x00	18.3.1.12/18-16
0x0_4503	ULCR—ULCR[DLAB] = x UART1 line control register	R/W	0x00	18.3.1.7/18-11
0x0_4504	UMCR—ULCR[DLAB] = x UART1 MODEM control register	R/W	0x00	18.3.1.8/18-13
0x0_4505	ULSR—ULCR[DLAB] = x UART1 line status register	R	0x60	18.3.1.9/18-14
0x0_4506	UMSR—ULCR[DLAB] = x UART1 MODEM status register	R	0x00	18.3.1.10/18-15
0x0_4507	USCR—ULCR[DLAB] = x UART1 scratch register	R/W	0x00	18.3.1.11/18-16
0x0_4510	UDSR—ULCR[DLAB] = x UART1 DMA status register	R	0x01	18.3.1.13/18-17
0x0_4600	URBR—ULCR[DLAB] = 0 UART2 receiver buffer register	R	0x00	18.3.1.1/18-6
0x0_4600	UTHR—ULCR[DLAB] = 0 UART2 transmitter holding register	W	0x00	18.3.1.2/18-6
0x0_4600	UDLB—ULCR[DLAB] = 1 UART2 divisor least significant byte register	R/W	0x00	18.3.1.3/18-7
0x0_4601	UIER—ULCR[DLAB] = 0 UART2 interrupt enable register	R/W	0x00	18.3.1.4/18-8
0x0_4601	UDMB_ULCR[DLAB] = 1 UART2 divisor most significant byte register	R/W	0x00	18.3.1.3/18-7
0x0_4602	UIIR—ULCR[DLAB] = 0 UART2 interrupt ID register	R	0x01	18.3.1.5/18-9
0x0_4602	UFCR—ULCR[DLAB] = 0 UART2 FIFO control register	W	0x00	18.3.1.6/18-10
0x0_4602	UAFR—ULCR[DLAB] = 1 UART2 alternate function register	R/W	0x00	18.3.1.12/18-16
0x0_4603	ULCR—ULCR[DLAB] = x UART2 line control register	R/W	0x00	18.3.1.7/18-11
0x0_4604	UMCR—ULCR[DLAB] = x UART2 MODEM control register	R/W	0x00	18.3.1.8/18-13
0x0_4605	ULSR—ULCR[DLAB] = x UART2 line status register	R	0x60	18.3.1.9/18-14
0x0_4606	UMSR—ULCR[DLAB] = x UART2 MODEM status register	R	0x00	18.3.1.10/18-15
0x0_4607	USCR—ULCR[DLAB] = x UART2 scratch register	R/W	0x00	18.3.1.11/18-16
0x0_4610	UDSR—ULCR[DLAB] = x UART2 DMA status register	R	0x01	18.3.1.13/18-17
0x0_4700–0x0_4FFF	Reserved, should be cleared	—	—	—

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
Local Bus Controller (LBC) Registers				
0x0_5000	BR0—Base register 0 **Note:** Port size for BR0 is configured from the value of RCWH[ROMLOC] which is loaded during reset, hence 'RR' is either 0x08, 0x10, or 0x18.	R/W	0x0000_RR01	10.3.1.1/10-11
0x0_5008	BR1—Base register 1		0x0000_0000	
0x0_5010	BR2—Base register 2			
0x0_5018	BR3—Base register 3			
0x0_5020	BR4—Base register 4			
0x0_5028	BR5—Base register 5			
0x0_5030	BR6—Base register 6			
0x0_5038	BR7—Base register 7			
0x0_5004	OR0—Options register 0	R/W	0x0000_0FF7	10.3.1.2/10-12
0x0_500C	OR1—Options register 1		0x0000_0000	
0x0_5014	OR2—Options register 2			
0x0_501C	OR3—Options register 3			
0x0_5024	OR4—Options register 4			
0x0_502C	OR5—Options register 5			
0x0_5034	OR6—Options register 6			
0x0_503C	OR7—Options register 7			
0x0_5068	MAR—UPM address register	R/W	0x0000_0000	10.3.1.3/10-18
0x0_506C	Reserved	—	—	—
0x0_5070	MAMR—UPMA mode register	R/W	0x0000_0000	10.3.1.4/10-19
0x0_5074	MBMR—UPMB mode register	R/W	0x0000_0000	10.3.1.4/10-19
0x0_5078	MCMR—UPMC mode register	R/W	0x0000_0000	10.3.1.4/10-19
0x0_507C– 0x0_5080	Reserved	—	—	—
0x0_5084	MRTPR—Memory refresh timer prescaler register	R/W	0x0000_0000	10.3.1.5/10-21
0x0_5088	MDR—UPM data register	R/W	0x0000_0000	10.3.1.6/10-22
0x0_508C– 0x0_5090	Reserved	—	—	—
0x0_5094	LSDMR—SDRAM mode register	R/W	0x0000_0000	10.3.1.7/10-22
0x0_50A0	LURT—UPM refresh timer	R/W	0x0000_0000	10.3.1.8/10-24
0x0_50A4	LSRT—SDRAM refresh timer	R/W	0x0000_0000	10.3.1.9/10-25
0x0_50B0	LTESR—Transfer error status register	Read/ bit-reset	0x0000_0000	10.3.1.10/10-26
0x0_50B4	LTEDR—Transfer error check disable register	R/W	0x0000_0000	10.3.1.11/10-27

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x0_50B8	LTEIR—Transfer error interrupt enable register	R/W	0x0000_0000	10.3.1.12/10-27
0x0_50BC	LTEATR—Transfer error attributes register	R/W	0x0000_0000	10.3.1.13/10-28
0x0_50C0	LTEAR—Transfer error address register	R/W	0x0000_0000	10.3.1.14/10-29
0x0_50C4–0x0_50CC	Reserved	—	—	—
0x0_50D0	LBCR—Local bus configuration register	R/W	0x0000_0000	10.3.1.15/10-29
0x0_50D4	LCRR—Clock ratio register	R/W	0x8000_0008	10.3.1.16/10-30
0x0_6000–0x0_6FFF	Reserved, should be cleared	—	—	—
Serial Peripheral Interface (SPI)				
0x0_7000–0x0_701F	Reserved, should be cleared	—	—	—
0x0_7020	SPMODE—SPI mode register	R/W	0x0000_0000	19.4.1.1/19-9
0x0_7024	SPIE—SPI event register	R/W	0x0000_0000	19.4.1.2/19-11
0x0_7028	SPIM—SPI mask register	R/W	0x0000_0000	19.4.1.3/19-13
0x0_702C	SPCOM—SPI command register	R/W	0x0000_0000	19.4.1.4/19-14
0x0_7030	SPITD—SPI transmit register	R/W	0x0000_0000	19.4.1.5/19-14
0x0_7034	SPIRD—SPI receive register	R	0xFFFF_FFFF	19.4.1.6/19-15
0x0_7038–0x0_7FFF	Reserved, should be cleared	—	—	—
DMA Registers				
0x0_8030	OMISR—Outbound message interrupt status register	Special	0x0000_0000	12.4.1/12-4
0x0_8034	OMIMR—Outbound message interrupt mask register	R/W	0x0000_0000	12.4.2/12-6
0x0_8050	IMR0—Inbound message register 0	R/W	0x0000_0000	12.4.3/12-7
0x0_8054	IMR1—Inbound message register 1	R/W	0x0000_0000	12.4.3/12-7
0x0_8058	OMR0—Outbound message register 0	R/W	0x0000_0000	12.4.4/12-7
0x0_805C	OMR1—Outbound message register 1	R/W	0x0000_0000	12.4.4/12-7
0x0_8060	ODR—Outbound doorbell register	R/W	0x0000_0000	12.4.5/12-8
0x0_8068	IDR—Inbound doorbell register	R/W	0x0000_0000	12.4.5/12-8
0x0_8080	IMISR—Inbound message interrupt status register	R/W	0x0000_0000	12.4.6/12-9
0x0_8084	IMIMR—Inbound message interrupt mask register	R/W	0x0000_0000	12.4.7/12-11
0x0_8100	DMAMR0—DMA 0 mode register	R/W	0x0000_0000	12.4.8.1/12-12
0x0_8104	DMASR0—DMA 0 status register	R/W	0x0000_0000	12.4.8.2/12-14
0x0_8108	DMACDAR0—DMA 0 current descriptor address register	R/W	0x0000_0000	12.4.8.3/12-15
0x0_8110	DMASAR0—DMA 0 source address register	R/W	0x0000_0000	12.4.8.4/12-16
0x0_8118	DMADAR0—DMA 0 destination address register	R/W	0x0000_0000	12.4.8.5/12-16

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x0_8120	DMABCR0—DMA 0 byte count register	R/W	0x0000_0000	12.4.8.6/12-17
0x0_8124	DMANDAR0—DMA 0 next descriptor address register	R/W	0x0000_0000	12.4.8.7/12-17
0x0_8180	DMAMR1—DMA 1 mode register	R/W	0x0000_0000	12.4.8.1/12-12
0x0_8184	DMASR1—DMA 1 status register	R/W	0x0000_0000	12.4.8.2/12-14
0x0_8188	DMACDAR1—DMA 1 current descriptor address register	R/W	0x0000_0000	12.4.8.3/12-15
0x0_8190	DMASAR1—DMA 1 source address register	R/W	0x0000_0000	12.4.8.4/12-16
0x0_8198	DMADAR1—DMA 1 destination address register	R/W	0x0000_0000	12.4.8.5/12-16
0x0_81A0	DMABCR1—DMA 1 byte count register	R/W	0x0000_0000	12.4.8.6/12-17
0x0_81A4	DMANDAR1—DMA 1 next descriptor address register	R/W	0x0000_0000	12.4.8.7/12-17
0x0_8200	DMAMR2—DMA 2 mode register	R/W	0x0000_0000	12.4.8.1/12-12
0x0_8204	DMASR2—DMA 2 status register	R/W	0x0000_0000	12.4.8.2/12-14
0x0_8208	DMACDAR2—DMA 2 current descriptor address register	R/W	0x0000_0000	12.4.8.3/12-15
0x0_8210	DMASAR2—DMA 2 source address register	R/W	0x0000_0000	12.4.8.4/12-16
0x0_8218	DMADAR2—DMA 2 destination address register	R/W	0x0000_0000	12.4.8.5/12-16
0x0_8220	DMABCR2—DMA 2 byte count register	R/W	0x0000_0000	12.4.8.6/12-17
0x0_8224	DMANDAR2—DMA 2 next descriptor address register	R/W	0x0000_0000	12.4.8.7/12-17
0x0_8280	DMAMR3—DMA 3 mode register	R/W	0x0000_0000	12.4.8.1/12-12
0x0_8284	DMASR3—DMA 3 status register	R/W	0x0000_0000	12.4.8.2/12-14
0x0_8288	DMACDAR3—DMA 3 current descriptor address register	R/W	0x0000_0000	12.4.8.3/12-15
0x0_8290	DMASAR3—DMA 3 source address register	R/W	0x0000_0000	12.4.8.4/12-16
0x0_8298	DMADAR3—DMA 3 destination address register	R/W	0x0000_0000	12.4.8.5/12-16
0x0_82A0	DMABCR3—DMA 3 byte count register	R/W	0x0000_0000	12.4.8.6/12-17
0x0_82A4	DMANDAR3—DMA 3 next descriptor address register	R/W	0x0000_0000	12.4.8.7/12-17
0x0_82A8	DMAGSR—DMA general status register	R	0x0000_0000	12.4.8.8/12-18
0x0_82B0–0x0_82FF	Reserved, should be cleared	—	—	—
PCI1 Software Configuration Registers				
0x0_8300	PCI1 CONFIG_ADDRESS	W	—	13.3.1.1/13-16
0x0_8304	PCI1 CONFIG_DATA	R/W	—	13.3.1.2/13-18
0x0_8308	PCI1 INT_ACK	R	—	13.3.1.3/13-18
0x0_830C–0x0_837F	Reserved, should be cleared	—	—	—
PCI2 Software Configuration Registers				
0x0_8380	PCI2 CONFIG_ADDRESS	W	—	13.3.1.1/13-16
0x0_8384	PCI2 CONFIG_DATA	R/W	—	13.3.1.2/13-18

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x0_8388	PCI2 INT_ACK	R	—	13.3.1.3/13-18
0x0_838C–0x0_83FF	Reserved, should be cleared	—	—	—
Sequencer (IOS)				
0x0_8400	POTAR0—PCI outbound translation address register 0	R/W	0x0000_0000	11.4.1/11-3
0x0_8408	POBAR0—PCI outbound base address register 0	R/W	0x0000_0000	11.4.2/11-3
0x0_8410	POCMR0—PCI outbound comparison mask register 0	R/W	0x0000_0000	11.4.3/11-4
0x0_8418	POTAR1—PCI outbound translation address register 1	R/W	0x0000_0000	11.4.1/11-3
0x0_8420	POBAR1—PCI outbound base address register 1	R/W	0x0000_0000	11.4.2/11-3
0x0_8428	POCMR1—PCI outbound comparison mask register 1	R/W	0x0000_0000	11.4.3/11-4
0x0_8430	POTAR2—PCI outbound translation address register 2	R/W	0x0000_0000	11.4.1/11-3
0x0_8438	POBAR2—PCI outbound base address register 2	R/W	0x0000_0000	11.4.2/11-3
0x0_8440	POCMR2—PCI outbound comparison mask register 2	R/W	0x0000_0000	11.4.3/11-4
0x0_8448	POTAR3—PCI outbound translation address register 3	R/W	0x0000_0000	11.4.1/11-3
0x0_8450	POBAR3—PCI outbound base address register 3	R/W	0x0000_0000	11.4.2/11-3
0x0_8458	POCMR3—PCI outbound comparison mask register 3	R/W	0x0000_0000	11.4.3/11-4
0x0_8460	POTAR4—PCI outbound translation address register 4	R/W	0x0000_0000	11.4.1/11-3
0x0_8468	POBAR4—PCI outbound base address register 4	R/W	0x0000_0000	11.4.2/11-3
0x0_8470	POCMR4—PCI outbound comparison mask register 4	R/W	0x0000_0000	11.4.3/11-4
0x0_8478	POTAR5—PCI outbound translation address register 5	R/W	0x0000_0000	11.4.1/11-3
0x0_8480	POBAR5—PCI outbound base address register 5	R/W	0x0000_0000	11.4.2/11-3
0x0_8488	POCMR5—PCI outbound comparison mask register 5	R/W	0x0000_0000	11.4.3/11-4
0x0_84F0	PMCR—Power management control register	R/W	0x0000_0000	11.4.4/11-5
0x0_84F8	DTCR—Discard timer control register	R/W	0x0000_0000	11.4.5/11-6
PCI1 Error Management Registers				
0x0_8500	PCI_ESR—PCI error status register	R / w1c	0x0000_0000	13.3.2.1/13-18
0x0_8504	PCI_ECDR—PCI error capture disable register	R/W	0x0000_0000	13.3.2.2/13-19
0x0_8508	PCI_EER—PCI error enable register	R/W	0x0000_0000	13.3.2.3/13-20
0x0_850C	PCI_EATCR—PCI error attributes capture register	R/W	0x0000_0000	13.3.2.4/13-21
0x0_8510	PCI_EACR—PCI error address capture register	R	0x0000_0000	13.3.2.5/13-23
0x0_8514	PCI_EEACR—PCI error extended address capture register	R	0x0000_0000	13.3.2.6/13-23
0x0_8518	PCI_EDLCR—PCI error data low capture register	R	0x0000_0000	13.3.2.7/13-24
0x0_851C	PCI_EDHCR—PCI error data high capture register	R	0x0000_0000	13.3.2.8/13-24

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
PCI1 Control and Status Registers				
0x0_8520	PCI_GCR—PCI general control register	R/W	0x0000_0000	13.3.2.9/13-24
0x0_8524	PCI_ECR—PCI error control register	R/W	0x0000_0000	13.3.2.10/13-25
0x0_8528	PCI_GSR—PCI general status register	R	0x0000_0000	13.3.2.11/13-26
PCI1 Inbound ATU Registers				
0x0_8538	PITAR2—PCI inbound translation address register 2	R/W	0x0000_0000	13.3.2.12/13-26
0x0_853C	Reserved, should be cleared	—	—	—
0x0_8540	PIBAR2—PCI inbound base address register 2	R/W	0x0000_0000	13.3.2.13/13-27
0x0_8544	PIEBAR2—PCI inbound extended base address register 2	R/W	0x0000_0000	13.3.2.14/13-27
0x0_8548	PIWAR2—PCI inbound window attributes register 2	R/W	0x0000_0000	13.3.2.15/13-28
0x0_8550	PITAR1—PCI inbound translation address register 1	R/W	0x0000_0000	13.3.2.12/13-26
0x0_8554	Reserved, should be cleared	—	—	—
0x0_8558	PIBAR1—PCI inbound base address register 1	R/W	0x0000_0000	13.3.2.13/13-27
0x0_855C	PIEBAR1—PCI inbound extended base address register 1	R/W	0x0000_0000	13.3.2.14/13-27
0x0_8560	PIWAR1—PCI inbound window attributes register 1	R/W	0x0000_0000	13.3.2.15/13-28
0x0_8568	PITAR0—PCI inbound translation address register 0	R/W	0x0000_0000	13.3.2.12/13-26
0x0_856C	Reserved, should be cleared	—	—	—
0x0_8570	PIBAR0—PCI inbound base address register 0	R/W	0x0000_0000	13.3.2.13/13-27
0x0_8578	PIWAR0—PCI inbound window attributes register 0	R/W	0x0000_0000	13.3.2.14/13-27
0x0_857C	Reserved, should be cleared	—	—	—
PCI2 Controller Control and Status Registers				
0x0_8600	PCI_ESR—PCI error status register	R / w1c	0x0000_0000	13.3.2.1/13-18
0x0_8604	PCI_ECDR—PCI error capture disable register	R/W	0x0000_0000	13.3.2.2/13-19
0x0_8608	PCI_EER—PCI error enable register	R/W	0x0000_0000	13.3.2.3/13-20
0x0_860C	PCI_EATCR—PCI error attributes capture register	R/W	0x0000_0000	13.3.2.4/13-21
0x0_8610	PCI_EACR—PCI error address capture register	R	0x0000_0000	13.3.2.5/13-23
0x0_8614	PCI_EEACR—PCI error extended address capture register	R	0x0000_0000	13.3.2.6/13-23
0x0_8618	PCI_EDLCR—PCI error data low capture register	R	0x0000_0000	13.3.2.7/13-24
0x0_861C	PCI_EDHCR—PCI error data high capture register	R	0x0000_0000	—
0x0_8620	PCI_GCR—PCI general control register	R/W	0x0000_0000	13.3.2.9/13-24
0x0_8624	PCI_ECR—PCI error control register	R/W	0x0000_0000	13.3.2.10/13-25
0x0_8628	PCI_GSR—PCI general status register	R	0x0000_0000	13.3.2.11/13-26
0x0_8638	PITAR2—PCI inbound translation address register 2	R/W	0x0000_0000	13.3.2.12/13-26
0x0_8640	PIBAR2—PCI inbound base address register 2	R/W	0x0000_0000	13.3.2.13/13-27
0x0_8644	PIEBAR2—PCI inbound extended base address register 2	R/W	0x0000_0000	13.3.2.14/13-27

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x0_8648	PIWAR2—PCI inbound window attributes register 2	R/W	0x0000_0000	13.3.2.15/13-28
0x0_8650	PITAR1—PCI inbound translation address register 1	R/W	0x0000_0000	13.3.2.12/13-26
0x0_8658	PIBAR1—PCI inbound base address register 1	R/W	0x0000_0000	13.3.2.13/13-27
0x0_865C	PIEBAR1—PCI inbound extended base address register 1	R/W	0x0000_0000	13.3.2.14/13-27
0x0_8660	PIWAR1—PCI inbound window attributes register 1	R/W	0x0000_0000	13.3.2.15/13-28
0x0_8668	PITAR0—PCI inbound translation address register 0	R/W	0x0000_0000	13.3.2.12/13-26
0x0_8670	PIBAR0—PCI inbound base address register 0	R/W	0x0000_0000	13.3.2.13/13-27
0x0_8678	PIWAR0—PCI inbound window attributes register 0	R/W	0x0000_0000	13.3.2.15/13-28
0x0_867C–0x0_86FF	Reserved, should be cleared	—	—	—
0x0_8700–0x2_1FFF	Reserved, should be cleared	—	—	—
USB MPH Controller Registers				
0x2_2000–0x2_20FF	Reserved, should be cleared	—	—	—
0x2_2100	CAPLENGTH—Capability register length	R	0x40	16.3.1.1/16-19
0x2_2102	HCIVERSION—Host interface version number	R	0x0100	16.3.1.2/16-19
0x2_2104	HCSPARAMS—Host crtl. structural parameters	R	0x0121_0012	16.3.1.3/16-20
0x2_2108	HCCPARAMS—Host crtl. capability parameters	R	0x0000_0006	16.3.1.4/16-21
0x2_2140	USBCMD—USB command	R/W	0x0008_*n*B*n*0	16.3.2.1/16-23
0x2_2144	USBSTS—USB status	R/W	0x0000_0000	16.3.2.2/16-26
0x2_2148	USBINTR—USB interrupt enable	R/W	0x0000_0000	16.3.2.3/16-28
0x2_214C	FRINDEX—USB frame index	R/W	0x0000_*nnnn*	16.3.2.4/16-30
0x2_2154	PERIODICLISTBASE—Frame list base address	R/W	0x*nnnn*_0000	16.3.2.6/16-31
0x2_2158	ASYNCLISTADDR—Next asynchronous list addr	R/W	0x0000_0000	16.3.2.8/16-32
0x2_215C	ASYNCTTSTS— Asynchronous buffer status for embedded TT	TBD	0x0000_0000	16.3.2.10/16-34
0x2_2160	BURSTSIZE—Programmable burst size	R/W	0x000_1010	16.3.2.11/16-34
0x2_2164	TXFILLTUNING—Host TT transmit pre-buffer packet tuning	R/W	0x0002_0000	16.3.2.12/16-35
0x2_2168	TXTTFILLTUNING—Host TT transmit pre-buffer packet tuning	R/W	0x0000_0000	16.3.2.13/16-37
0x2_2170	ULPI VIEWPORT—ULPI Register Access	R/W	0x0000_0000	16.3.2.14/16-37
0x2_2180	CONFIGFLAG—Configured flag register	R	0x0000_0001	16.3.2.15/16-39
0x2_2184	PORTSC1—Port status/control 1	R/W	0x8C00_0001	16.3.2.16/16-39
0x2_2188	PORTSC2—Port status/control 2	R/W	0x8C00_0001	16.3.2.16/16-39
0x2_21A8	USBMODE—USB device mode	R/W	0x0000_0003	16.3.2.18/16-47
0x2_2400	SNOOP1—Snoop 1	R/W	0x0000_0000	16.3.2.26/16-53
0x2_2404	SNOOP2—Snoop 2	R/W	0x0000_0000	16.3.2.26/16-53

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x2_2408	AGE_CNT_THRESH—Age count threshold	R/W	0x0000_0000	16.3.2.27/16-54
0x2_240C	SI_CTRL—System interface control	R/W	0x0000_0000	16.3.2.28/16-56
0x2_2410	PRI_CTRL—Priority control	R/W	0x0000_0000	16.3.2.29/16-56
0x2_2500	CONTROL—Control	R/W	0x0000_0000	16.3.2.30/16-57
0x2_2504–0x2_2FFF	Reserved, should be cleared	—	—	—
USB DR Controller Registers				
0x2_3000–0x2_30FF	Reserved, should be cleared	—	—	—
0x2_3100	CAPLENGTH—Capability register length	R	0x40	16.3.1.1/16-19
0x2_3102	HCIVERSION—Host interface version number	R	0x0100	16.3.1.2/16-19
0x2_3104	HCSPARAMS—Host crtl. structural parameters	R	0x0111_0011	16.3.1.3/16-20
0x2_3108	HCCPARAMS—Host crtl. capability parameters	R	0x0000_0006	16.3.1.4/16-21
0x2_3120	DCIVERSION—Device interface version number	R	0x0001	16.3.1.5/16-22
0x2_3124	DCCPARAMS—Device controller parameters	R	0x0000_0186	16.3.1.6/16-22
0x2_3140	USBCMD—USB command	R/W	0x0008_*n*B*n*0	16.3.2.1/16-23
0x2_3144	USBSTS—USB status	R/W	0x0000_0000	16.3.2.2/16-26
0x2_3148	USBINTR—USB interrupt enable	R/W	0x0000_0000	16.3.2.3/16-28
0x2_314C	FRINDEX—USB frame index	R/W	0x0000_*nnnn*	16.3.2.4/16-30
0x2_3154	PERIODICLISTBASE—Frame list base address	R/W	0x*nnnn*_0000	16.3.2.6/16-31
0x2_3154	DEVICEADDR—USB device address	R/W	0x0000_0000	16.3.2.7/16-32
0x2_3158	ASYNCLISTADDR—Next asynchronous list addr (host mode)	R/W	0x0000_0000	16.3.2.8/16-32
0x2_3158	ENDPOINT ADDR—Address at endpoint list (device mode)	R/W	0x0000_0000	16.3.2.9/16-33
0x2_3160	BURSTSIZE—Programmable burst size	R/W	0x0000_1010	16.3.2.11/16-34
0x2_3164	TXFILLTUNING—Host TT transmit pre-buffer packet tuning	R/W	0x0002_0000	16.3.2.12/16-35
0x2_3170	ULPI VIEWPORT—ULPI Register Access	R/W	0x0000_0000	16.3.2.14/16-37
0x2_3180	CONFIGFLAG—Configured flag register	R	0x0000_0001	16.3.2.15/16-39
0x2_3184	PORTSC1—Port status/control	R/W	0x9C00_0000	16.3.2.16/16-39
0x2_31A4	OTGSC—On-the-Go status and control	R/W	0x0000_0001	16.3.2.17/16-44
0x2_31A8	USBMODE—USB device mode	R/W	0x0000_0000	16.3.2.18/16-47
0x2_31AC	ENDPTSETUPSTAT—Endpoint setup status	R/W	0x0000_0000	16.3.2.19/16-48
0x2_31B0	ENDPOINTPRIME—Endpoint initialization	R/W	0x0000_0000	16.3.2.20/16-48
0x2_31B4	ENDPTFLUSH—Endpoint de-initialize	R/W	0x0000_0000	16.3.2.21/16-49
0x2_31B8	ENDPTSTATUS—Endpoint status	R	0x0000_0000	16.3.2.22/16-50
0x2_31BC	ENDPTCOMPLETE—Endpoint complete	R/W	0x0000_0000	16.3.2.23/16-50

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x2_31C0	ENDPTCTRL0—Endpoint control 0	R/W	0x0080_0080	16.3.2.24/16-51
0x2_31C4	ENDPTCTRL1—Endpoint control 1	R/W	0x0000_0000	16.3.2.25/16-52
0x2_31C8	ENDPTCTRL2—Endpoint control 2	R/W	0x0000_0000	16.3.2.25/16-52
0x2_31CA	ENDPTCTRL3—Endpoint control 3	R/W	0x0000_0000	16.3.2.25/16-52
0x2_31D0	ENDPTCTRL4—Endpoint control 4	R/W	0x0000_0000	16.3.2.25/16-52
0x2_31D4	ENDPTCTRL5—Endpoint control 5	R/W	0x0000_0000	16.3.2.25/16-52
0x2_3400	SNOOP1—Snoop 1	R/W	0x0000_0000	16.3.2.26/16-53
0x2_3404	SNOOP2—Snoop 2	R/W	0x0000_0000	16.3.2.26/16-53
0x2_3408	AGE_CNT_THRESH—Age count threshold	R/W	0x0000_0000	16.3.2.27/16-54
0x2_340C	PRI_CTRL—Priority control	R/W	0x0000_0000	16.3.2.29/16-56
0x2_3410	SI_CTRL—System interface control	R/W	0x0000_0000	16.3.2.28/16-56
0x2_3500	CONTROL—Control	R/W	0x0000_0000	16.3.2.30/16-57
0x2_3504–0x2_3FFF	Reserved, should be cleared	—	—	—
TSEC1 General Control and Status Registers				
0x2_4000–0x2_400C	Reserved, should be cleared	—	—	—
0x2_4010	IEVENT—Interrupt event register	R/W	0x0000_0000	15.5.3.1.1/15-19
0x2_4014	IMASK—Interrupt mask register	R/W	0x0000_0000	15.5.3.1.2/15-22
0x2_4018	EDIS—Error disabled register	R/W	0x0000_0000	15.5.3.1.3/15-24
0x2_401C	Reserved, should be cleared	—	—	—
0x2_4020	ECNTRL—Ethernet control register	R/W	0x0000_0000	15.5.3.1.4/15-25
0x2_4024	MINFLR—Minimum frame length register	R/W	0x0000_0040	15.5.3.1.5/15-26
0x2_4028	PTV—Pause time value register	R/W	0x0000_0000	15.5.3.1.6/15-27
0x2_402C	DMACTRL—DMA control register	R/W	0x0000_0000	15.5.3.1.7/15-28
0x2_4030	TBIPA—TBI PHY address register	R/W	0x0000_0000	15.5.3.1.8/15-29
0x2_4034–0x2_4038	Reserved, should be cleared	—	—	—
TSEC1 FIFO Control and Status Registers				
0x2_4040–0x2_4088	Reserved, should be cleared	R	0x0000_0000	—
0x2_408C	FIFO_TX_THR—FIFO transmit threshold register	R/W	0x0000_0100	15.5.3.2.1/15-30
0x2_4090	Reserved, should be cleared	R	0x0000_0000	—
0x2_4094	FIFO_TX_SP—FIFO transmit space available register	R/W	0x0000_0010	15.5.3.2.2/15-31
0x2_4098	FIFO_TX_STARVE—FIFO transmit starve register	R/W	0x0000_0080	15.5.3.2.3/15-31

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x2_409C	FIFO_TX_STARVE_SHUTOFF—FIFO transmit starve shutoff register	R/W	0x0000_0100	15.5.3.2.4/15-32
0x2_40A0–0x2_40FC	Reserved, should be cleared	R	0x0000_0000	—
TSEC1 Transmit Control and Status Registers				
0x2_4100	TCTRL—Transmit control register	R/W	0x0000_0000	15.5.3.3.1/15-33
0x2_4104	TSTAT—Transmit status register	R/W	0x0000_0000	15.5.3.3.2/15-34
0x2_4108	Reserved, should be cleared	R	0x0000_0000	—
0x2_4110	TXIC—Transmit interrupt coalescing configuration register	R/W	0x0000_0000	
0x2_4114–0x2_4120	Reserved, should be cleared	—	—	—
0x2_4124	CTBPTR—Current TxBD pointer register	R	0x0000_0000	15.5.3.3.5/15-36
0x2_4128–0x2_4180	Reserved, should be cleared	—	—	—
0x2_4184	TBPTR—TxBD pointer register	R/W	0x0000_0000	15.5.3.3.6/15-36
0x2_4188–0x2_4200	Reserved, should be cleared	—	—	—
0x2_4204	TBASE—TxBD base address register	R/W	0x0000_0000	15.5.3.3.7/15-37
0x2_4208–0x2_42AC	Reserved, should be cleared	—	—	—
0x2_42B0	OSTBD—Out-of-sequence TxBD register	R/W	0x0800_0000	15.5.3.3.8/15-37
0x2_42B4	OSTBDP—Out-of-sequence Tx data buffer pointer register	R/W	0x0000_0000	15.5.3.3.9/15-39
0x2_42B8–0x2_42FC	Reserved, should be cleared	—	—	—
TSEC1 Receive Control and Status Registers				
0x2_4300	RCTRL—Receive control register	R/W	0x0000_0000	15.5.3.4.1/15-40
0x2_4304	RSTAT—Receive status register	R/W	0x0000_0000	15.5.3.4.2/15-41
0x2_4308	Reserved, should be cleared	R	0x0000_0000	—
0x2_430C	RBDLEN—RxBD data length register	R	0x0000_0000	15.5.3.4.3/15-41
0x2_4310	RXIC—Receive interrupt coalescing configuration register	R/W	0x0000_0000	15.5.3.4.4/15-42
0x2_4314–0x2_4320	Reserved, should be cleared	—	—	—
0x2_4324	CRBPTR—Current RxBD pointer register	R	0x0000_0000	15.5.3.4.5/15-43
0x2_4328–0x2_433C	Reserved, should be cleared	—	—	—
0x2_4340	MRBLR—Maximum receive buffer length register	R/W	0x0000_0000	15.5.3.4.6/15-43
0x2_4344–0x2_4380	Reserved, should be cleared	—	—	—

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x2_4384	RBPTR—RxBD pointer register	R/W	0x0000_0000	15.5.3.4.7/15-44
0x2_4388–0x2_4400	Reserved, should be cleared	—	—	—
0x2_4404	RBASE—RxBD base address register	R/W	0x0000_0000	15.5.3.4.8/15-44
0x2_4408–0x2_44FC	Reserved, should be cleared	—	—	—
TSEC1 MAC Registers				
0x2_4500	MACCFG1—MAC configuration register 1	R/W, R	0x0000_0000	15.5.3.6.1/15-48
0x2_4504	MACCFG2—MAC configuration register 2	R/W	0x0000_7000	15.5.3.6.2/15-49
0x2_4508	IPGIFG—Inter-packet gap/inter-frame gap register	R/W	0x4060_5060	15.5.3.6.3/15-51
0x2_450C	HAFDUP—Half-duplex register	R/W	0x00A1_F037	15.5.3.6.4/15-52
0x2_4510	MAXFRM—Maximum frame length register	R/W	0x0000_0600	15.5.3.6.5/15-53
0x2_4514–0x2_451C	Reserved, should be cleared	—	—	—
0x2_4520	MIIMCFG—MII management configuration register	R/W	0x0000_0000	15.5.3.6.6/15-53
0x2_4524	MIIMCOM—MII management command register	R/W	0x0000_0000	15.5.3.6.7/15-54
0x2_4528	MIIMADD—MII management address register	R/W	0x0000_0000	15.5.3.6.8/15-55
0x2_452C	MIIMCON—MII management control register	W	0x0000_0000	15.5.3.6.9/15-56
0x2_4530	MIIMSTAT—MII management status register	R	0x0000_0000	15.5.3.6.10/15-56
0x2_4534	MIIMIND—MII management indicator register	R	0x0000_0000	15.5.3.6.11/15-57
0x2_4538	Reserved, should be cleared	—	—	—
0x2_453C	IFSTAT—Interface status register	Special	0x0000_0001	15.5.3.6.12/15-58
0x2_4540	MACSTNADDR1—Station address register, part 1	R/W	0x0000_0000	15.5.3.6.13/15-58
0x2_4544	MACSTNADDR2—Station address register, part 2	R/W	0x0000_0000	15.5.3.6.14/15-59
0x2_4548–0x2_467C	Reserved, should be cleared	—	—	—
TSEC1 RMON MIB Registers				
TSEC1 Transmit and Receive Counters				
0x2_4680	TR64—Transmit and receive 64-byte frame counter register	R/W	0x0000_0000	15.5.3.7.1/15-60
0x2_4684	TR127—Transmit and receive 65- to 127-byte frame counter register	R/W	0x0000_0000	15.5.3.7.2/15-61
0x2_4688	TR255—Transmit and receive 128- to 255-byte frame counter register	R/W	0x0000_0000	15.5.3.7.3/15-61
0x2_468C	TR511—Transmit and receive 256- to 511-byte frame counter register	R/W	0x0000_0000	15.5.3.7.4/15-62
0x2_4690	TR1K—Transmit and receive 512- to 1023-byte frame counter register	R/W	0x0000_0000	15.5.3.7.5/15-62

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x2_4694	TRMAX—Transmit and receive 1024- to 1518-byte frame counter register	R/W	0x0000_0000	15.5.3.7.6/15-63
0x2_4698	TRMGV—Transmit and receive 1519- to 1522-byte good VLAN frame count register	R/W	0x0000_0000	15.5.3.7.7/15-63
TSEC1 Receive Counters				
0x2_469C	RBYT—Receive byte counter register	R/W	0x0000_0000	15.5.3.7.8/15-64
0x2_46A0	RPKT—Receive packet counter register	R/W	0x0000_0000	15.5.3.7.9/15-64
0x2_46A4	RFCS—Receive FCS error counter register	R/W	0x0000_0000	15.5.3.7.10/15-65
0x2_46A8	RMCA—Receive multicast packet counter register	R/W	0x0000_0000	15.5.3.7.11/15-65
0x2_46AC	RBCA—Receive broadcast packet counter register	R/W	0x0000_0000	15.5.3.7.12/15-66
0x2_46B0	RXCF—Receive control frame packet counter register	R/W	0x0000_0000	15.5.3.7.13/15-66
0x2_46B4	RXPF—Receive PAUSE frame packet counter register	R/W	0x0000_0000	15.5.3.7.14/15-67
0x2_46B8	RXUO—Receive unknown OP code counter register	R/W	0x0000_0000	15.5.3.7.15/15-67
0x2_46BC	RALN—Receive alignment error counter register	R/W	0x0000_0000	15.5.3.7.16/15-68
0x2_46C0	RFLR—Receive frame length error counter register	R/W	0x0000_0000	15.5.3.7.17/15-68
0x2_46C4	RCDE—Receive code error counter register	R/W	0x0000_0000	15.5.3.7.18/15-69
0x2_46C8	RCSE—Receive carrier sense error counter register	R/W	0x0000_0000	15.5.3.7.19/15-69
0x2_46CC	RUND—Receive undersize packet counter register	R/W	0x0000_0000	15.5.3.7.20/15-70
0x2_46D0	ROVR—Receive oversize packet counter register	R/W	0x0000_0000	15.5.3.7.21/15-70
0x2_46D4	RFRG—Receive fragments counter register	R/W	0x0000_0000	15.5.3.7.22/15-71
0x2_46D8	RJBR—Receive jabber counter register	R/W	0x0000_0000	15.5.3.7.23/15-71
0x2_46DC	RDRP—Receive drop register	R/W	0x0000_0000	15.5.3.7.24/15-72
TSEC1 Transmit Counters				
0x2_46E0	TBYT—Transmit byte counter register	R/W	0x0000_0000	15.5.3.7.25/15-72
0x2_46E4	TPKT—Transmit packet counter register	R/W	0x0000_0000	15.5.3.7.26/15-73
0x2_46E8	TMCA—Transmit multicast packet counter register	R/W	0x0000_0000	15.5.3.7.27/15-73
0x2_46EC	TBCA—Transmit broadcast packet counter register	R/W	0x0000_0000	15.5.3.7.28/15-74
0x2_46F0	TXPF—Transmit PAUSE control frame counter register	R/W	0x0000_0000	15.5.3.7.29/15-74
0x2_46F4	TDFR—Transmit deferral packet counter register	R/W	0x0000_0000	15.5.3.7.30/15-75
0x2_46F8	TEDF—Transmit excessive deferral packet counter register	R/W	0x0000_0000	15.5.3.7.31/15-75
0x2_46FC	TSCL—Transmit single collision packet counter register	R/W	0x0000_0000	15.5.3.7.32/15-76
0x2_4700	TMCL—Transmit multiple collision packet counter register	R/W	0x0000_0000	15.5.3.7.33/15-76
0x2_4704	TLCL—Transmit late collision packet counter register	R/W	0x0000_0000	15.5.3.7.34/15-77
0x2_4708	TXCL—Transmit excessive collision packet counter register	R/W	0x0000_0000	15.5.3.7.35/15-77
0x2_470C	TNCL—Transmit total collision counter register	R/W	0x0000_0000	15.5.3.7.36/15-78

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x2_4710	Reserved, should be cleared	R	0x0000_0000	—
0x2_4714	TDRP—Transmit drop frame counter register	R/W	0x0000_0000	15.5.3.7.37/15-78
0x2_4718	TJBR—Transmit jabber frame counter register	R/W	0x0000_0000	15.5.3.7.38/15-79
0x2_471C	TFCS—Transmit FCS error counter register	R/W	0x0000_0000	15.5.3.7.39/15-79
0x2_4720	TXCF—Transmit control frame counter register	R/W	0x0000_0000	15.5.3.7.40/15-80
0x2_4724	TOVR—Transmit oversize frame counter register	R/W	0x0000_0000	15.5.3.7.41/15-80
0x2_4728	TUND—Transmit undersize frame counter register	R/W	0x0000_0000	15.5.3.7.42/15-81
0x2_472C	TFRG—Transmit fragments frame counter register	R/W	0x0000_0000	15.5.3.7.43/15-81
TSEC1 General Registers				
0x2_4730	CAR1—Carry register one register	R	0x0000_0000	15.5.3.7.44/15-82
0x2_4734	CAR2—Carry register two register	R	0x0000_0000	15.5.3.7.45/15-83
0x2_4738	CAM1—Carry register one mask register	R/W	0xFE01_FFFF	15.5.3.7.46/15-85
0x2_473C	CAM2—Carry register two mask register	R/W	0x000F_FFFF	15.5.3.7.47/15-86
0x2_4740–0x2_47FC	Reserved, should be cleared	—	—	—
TSEC1 Hash Function Registers				
0x2_4800	IADDR0—Individual address register 0	R/W	0x0000_0000	15.5.3.8.1/15-87
0x2_4804	IADDR1—Individual address register 1	R/W	0x0000_0000	
0x2_4808	IADDR2—Individual address register 2	R/W	0x0000_0000	
0x2_480C	IADDR3—Individual address register 3	R/W	0x0000_0000	
0x2_4810	IADDR4—Individual address register 4	R/W	0x0000_0000	
0x2_4814	IADDR5—Individual address register 5	R/W	0x0000_0000	
0x2_4818	IADDR6—Individual address register 6	R/W	0x0000_0000	
0x2_481C	IADDR7—Individual address register 7	R/W	0x0000_0000	
0x2_4820–0x2_487C	Reserved, should be cleared	—	—	—
0x2_4880	GADDR0—Group address register 0	R/W	0x0000_0000	15.5.3.8.2/15-88
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	

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x2_48A0–0x2_4AFF	Reserved, should be cleared	—	—	—
TSEC1 Attribute Registers				
0x2_4B00–0x2_4BF4	Reserved, should be cleared	—	—	—
0x2_4BF8	ATTR—Attribute register	R	0x0000_0000	
0x2_4BFC	ATTRELI—Attribute extract length and extract index register	R/W	0x0000_0000	
0x2_48A0–0x2_4BFC	Reserved, should be cleared	—	—	—
TSEC1 Future Expansion Space				
0x2_4C00–0x2_4FFF	Reserved, should be cleared	—	—	—
TSEC2 Registers				
0x2_5000–0x2_5FFF	TSEC2 registers **Note:** TSEC2 has the same memory-mapped registers that are described for TSEC1 from 0x 2_4000 to 0x2_4FFF except the offsets are from 0x 2_5000 to 0x2_5FFF.			
Security Engine Address Map Registers				
Controller Registers				
0x3_0000–0x3_0FFF	Reserved, should be cleared	—	—	—
0x3_1008	IMR—Interrupt mask register	R/W	0x0000_0000_0000_0000	14.7.2.1/14-94
0x3_1010	ISR—Interrupt status register	R	0x0000_0000_0000_0000	14.7.2.2/14-96
0x3_1018	ICR—Interrupt clear register	W	0x0000_0000_0000_0000	14.7.2.3/14-96
0x3_1020	ID—Identification register	R	0x0000_0000_0000_00400x	14.7.2.4/14-98
0x3_1028	EUASR—EU assignment status register	R	0xF0F0_F0F0_00FF_F0F0	14.7.2/14-93
0x3_1030	MCR—Master control register	R/W	0000_0000_0000_0000	14.7.2.5/14-98
Channel 1				
0x3_1108	CCCR1—Crypto-channel 1 configuration register	R/W	0x0000_0000_0000_0000	14.6.1.1/14-82
0x3_1110	CCPSR1—Crypto-channel 1 pointer status register	R	0x0000_0000_0000_0007	14.6.1.2/14-85
0x3_1140	CDPR1—Crypto-channel 1 current descriptor pointer register	R	0x0000_0000_0000_0000	14.6.1.3/14-90

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x3_1148	FF1—Crypto-channel 1 fetch FIFO address register	W	0x0000_0000 _0000_0000	14.6.1.4/14-90
0x3_1180–0x3_11BF	DB*n*—Crypto-channel 1 descriptor buffers [0–7]	R	0x0000_0000 _0000_0000	14.6.1.5/14-91
Channel 2				
0x3_1208	CCCR2—Crypto-channel 2 configuration register	R/W	0x0000_0000 _0000_0000	14.6.1.1/14-82
0x3_1210	CCPSR2—Crypto-channel 2 pointer status register	R	0x0000_0000 _0000_0007	14.6.1.2/14-85
0x3_1240	CDPR2—Crypto-channel 2 current descriptor pointer register	R	0x0000_0000 _0000_0000	14.6.1.3/14-90
0x3_1248	FF2—Crypto-channel 2 fetch FIFO address register	W	0x0000_0000 _0000_0000	14.6.1.4/14-90
0x3_1280–0x3_12BF	DBn—Crypto-channel 2 descriptor buffers[0–7]	R	0x0000_0000 _0000_0000	14.6.1.5/14-91
Channel 3				
0x3_1308	CCCR3—Crypto-channel 3 configuration register	R/W	0x0000_0000 _0000_0000	14.6.1.1/14-82
0x3_1310	CCPSR3—Crypto-channel 3 pointer status register	R	0x0000_0000 _0000_0007	14.6.1.2/14-85
0x3_1340	CDPR3—Crypto-channel 3 current descriptor pointer register	R	0x0000_0000 _0000_0000	14.6.1.3/14-90
0x3_1348	FF3—Crypto-channel 3 fetch FIFO address register	W	0x0000_0000 _0000_0000	14.6.1.4/14-90
0x3_1380–0x3_13BF	DBn—Crypto-channel 3 descriptor buffers[0–7]	R	0x0000_0000 _0000_0000	14.6.1.5/14-91
Channel 4				
0x3_1408	CCCR4—Crypto-channel 4 configuration register	R/W	0x0000_0000 _0000_0000	14.6.1.1/14-82
0x3_1410	CCPSR4—Crypto-channel 4 pointer status register	R	0x0000_0000 _0000_0007	14.6.1.2/14-85
0x3_1440	CDPR4—Crypto-channel 4 current descriptor pointer register	R	0x0000_0000 _0000_0000	14.6.1.3/14-90
0x3_1448	FF4—Crypto-channel 4 fetch FIFO address register	W	0x0000_0000 _0000_0000	14.6.1.4/14-90
0x3_1480–0x3_14BF	DBn—Crypto-channel 4 descriptor buffers[0–7]	R	0x0000_0000 _0000_0000	14.6.1.5/14-91
Data Encryption Standard Execution Unit (DEU)				
0x3_2000	DEUMR—DEU mode register	R/W	0x0000_0000 _0000_0000	14.5.2.1/14-35

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x3_2008	DEUKSR—DEU key size register	R/W	0x0000_0000 _0000_0000	14.5.2.2/14-36
0x3_2010	DEUDSR—DEU data size register	R/W	0x0000_0000 _0000_0000	14.5.2.3/14-36
0x3_2018	DEURCR—DEU reset control register	R/W	0x0000_0000 _0000_0000	14.5.2.4/14-37
0x3_2028	DEUSR—DEU status register	R	0x0000_0000 _0000_0000	14.5.2.5/14-37
0x3_2030	DEUISR—DEU interrupt status register	R	0x0000_0000 _0000_0000	14.5.2.6/14-38
0x3_2038	DEUICR—DEU interrupt control register	R/W	0x0000_0000 _0000_3000	14.5.2.7/14-40
0x3_2050	DEUEUG—DEU EU-Go register	W	0x0000_0000 _0000_0000	14.5.2.8/14-41
0x3_2100	DEUIV—DEU initialization vector register	R/W	0x0000_0000 _0000_0000	14.5.2.9/14-42
0x3_2400	DEUK1—DEU key 1 register	W	—	14.5.2.10/14-42
0x3_2408	DEUK2—DEU key 2 register	W	—	14.5.2.10/14-42
0x3_2410	DEUK3—DEU key 3 register	W	—	14.5.2.10/14-42
0x3_2800–0x3_2FFF	DEU FIFO	R/W	0x0000_0000 _0000_0000	14.5.2.11/14-42
Advanced Encryption Standard Execution Unit (AESU)				
0x3_4000	AESUMR—AESU mode register	R/W	0x0000_0000 _0000_0000	14.5.6.1/14-68
0x3_4008	AESUKSR—AESU key size register	R/W	0x0000_0000 _0000_0000	14.5.6.2/14-71
0x3_4010	AESUDSR—AESU data size register	R/W	0x0000_0000 _0000_0000	14.5.6.3/14-71
0x3_4018	AESURCR—AESU reset control register	R/W	0x0000_0000 _0000_0000	14.5.6.4/14-72
0x3_4028	AESUSR—AESU status register	R	0x0000_0000 _0000_0000	14.5.6.5/14-73
0x3_4030	AESUISR—AESU interrupt status register	R	0x0000_0000 _0000_0000	14.5.6.6/14-74
0x3_4038	AESUICR—AESU interrupt control register	R/W	0x0000_0000 _0000_1000	14.5.6.7/14-75
0x3_4050	AESUEMR—AESU end-of-message register	W	0x0000_0000 _0000_0000	14.5.6.8/14-76
0x3_4100	AESU context memory registers	R/W	0x0000_0000 _0000_0000	14.5.6.9/14-77

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x3_4400–0x3_4408	AESU key memory	R/W	0x0000_0000_0000_0000	14.5.6.9.5/14-81
0x3_4800–0x3_4FFF	AESU FIFO	R/W	0x0000_0000_0000_0000	14.5.6.9.6/14-81
Message Digest Execution Unit (MDEU)				
0x3_6000	MDEUMR—MDEU mode register	R/W	0x0000_0000_0000_0000	14.5.4.1/14-51
0x3_6008	MDEUKSR—MDEU key size register	R/W	0x0000_0000_0000_0000	14.5.4.3/14-55
0x3_6010	MDEUDSR—MDEU data size register	R/W	0x0000_0000_0000_0000	14.5.4.4/14-56
0x3_6018	MDEURCR—MDEU reset control register	R/W	0x0000_0000_0000_0000	14.5.4.5/14-56
0x3_6028	MDEUSR—MDEU status register	R	0x0000_0000_0000_0000	14.5.4.6/14-57
0x3_6030	MDEUISR—MDEU interrupt status register	R	0x0000_0000_0000_0000	14.5.4.7/14-58
0x3_6038	MDEUICR—MDEU interrupt control register	R/W	0x0000_0000_0000_1000	14.5.4.8/14-59
0x3_6050	MDEUEUG—MDEU EU-Go register	W	0x0000_0000_0000_0000	14.5.4.10/14-61
0x3_6100–0x3_6120	MDEU context memory registers	R/W	0x0000_0000_0000_0000	14.5.4.11/14-61
0x3_6400–0x3_647F	MDEU key memory	W	0x0000_0000_0000_0000	14.5.4.12/14-62
0x3_6800–0x3_6FFF	MDEU FIFO	W	0x0000_0000_0000_0000	14.5.4.13/14-63
ARC Four Execution Unit (AFEU)				
0x3_8000	AFEUMR—AFEU mode register	R/W	0x0000_0000_0000_0000	14.5.3.1/14-43
0x3_8008	AFEUKSR—AFEU key size register	R/W	0x0000_0000_0000_0000	14.5.3.3/14-44
0x3_8010	AFEUDSR—AFEU data size register	R/W	0x0000_0000_0000_0000	14.5.3.4/14-45
0x3_8018	AFEURCR—AFEU reset control register	R/W	0x0000_0000_0000_0000	14.5.3.5/14-46
0x3_8028	AFEUSR—AFEU status register	R	0x0000_0000_0000_0000	14.5.3.6/14-46
0x3_8030	AFEUISR—AFEU interrupt status register	R	0x0000_0000_0000_0000	14.5.3.7/14-47

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x3_8038	AFEUICR—AFEU interrupt control register	R/W	0x0000_0000 _0000_1000	14.5.3.8/14-49
0x3_8050	AFEUEMR—AFEU end of message register	W	0x0000_0000 _0000_0000	14.5.3.9/14-50
0x3_8100–0x3_81FF	AFEU context memory registers	R/W	0x0000_0000 _0000_0000	14.5.3.10.1/14-50
0x3_8200	AFEU context memory pointers	R/W	0x0000_0000 _0000_0000	14.5.3.10.2/14-51
0x3_8400	AFEUK0—AFEU key register 0	W	—	14.5.3.11/14-51
0x3_848	AFEUK1—AFEU key register 1	W	—	14.5.3.11/14-51
0x3_8800–0x3_8FFF	AFEU FIFO	R/W	0x0000_0000 _0000_0000	14.5.3.11.1/14-51
Random Number Generator (RNG)				
0x3_A000	RNGMR—RNG mode register	R/W	0x0000_0000 _0000_0000	14.5.5.1/14-63
0x3_A010	RNGDSR—RNG data size register	R/W	0x0000_0000 _0000_0000	14.5.5.2/14-64
0x3_A018	RNGRCR—RNG reset control register	R/W	0x0000_0000 _0000_0000	14.5.5.3/14-65
0x3_A028	RNGSR—RNG status register	R	0x0000_0000 _0000_0000	14.5.5.4/14-65
0x3_A030	RNGISR—RNG interrupt status register	R	0x0000_0000 _0000_0000	14.5.5.5/14-66
0x3_A038	RNGICR—RNG interrupt control register	R/W	0x0000_0000 _0000_1000	14.5.5.6/14-67
0x3_A050	RNGEUG—RNG EU-Go register	W	0x0000_0000 _0000_0000	14.5.5.7/14-68
0x3_A800–0x3_AFFF	RNG FIFO	R	0x0000_0000 _0000_0000	14.5.5.8/14-68
Public Key Execution Unit (PKEU)				
0x3_C000	PKEUMR—PKEU mode register	R/W	0x0000_0000 _0000_0000	14.5.1.1/14-26
0x3_C008	PKEUKSR—PKEU key size register	R/W	0x0000_0000 _0000_0000	14.5.1.2/14-28
0x3_C010	PKEUDSR—PKEU data size register	R/W	0x0000_0000 _0000_0000	14.5.1.3/14-28
0x3_C018	PKEURCR—PKEU reset control register	R/W	0x0000_0000 _0000_0000	14.5.1.5/14-29
0x3_C028	PKEUSR—PKEU status register	R	0x0000_0000 _0000_0000	14.5.1.6/14-30

Table 2-2. Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x3_C030	PKEUISR—PKEU interrupt status register	R	0x0000_0000 _0000_0000	14.5.1.7/14-31
0x3_C038	PKEUICR—PKEU interrupt control register	R/W	0x0000_0000 _0000_1000	14.5.1.8/14-32
0x3_C040	PKEUABS—PKEU AB size register	R/W	0x0000_0000 _0000_0000	14.5.1.3/14-28
0x3_C050	PKEUEUG—PKEU EU-Go	W	0x0000_0000 _0000_0000	14.5.1.9/14-33
0x3_C200–0x3_C23F	PKEU parameter memory A0	R/W	0x0000_0000 _0000_0000	14.5.1.10/14-34
0x3_C240–0x3_C27F	PKEU parameter memory A1	R/W	0x0000_0000 _0000_0000	
0x3_C280–0x3_C2BF	PKEU parameter memory A2	R/W	0x0000_0000 _0000_0000	
0x3_C2C0–0x3_C2FF	PKEU parameter memory A3	R/W	0x0000_0000 _0000_0000	
0x3_C300–0x3_C33F	PKEU parameter memory B0	R/W	0x0000_0000 _0000_0000	
0x3_C340–0x3_C37F	PKEU parameter memory B1	R/W	0x0000_0000 _0000_0000	
0x3_C380–0x3_C3BF	PKEU parameter memory B2	R/W	0x0000_0000 _0000_0000	
0x3_C3C0–0x3_C3FF	PKEU parameter memory B3	R/W	0x0000_0000 _0000_0000	
0x3_C400–0x3_C4FF	PKEU parameter memory E	W	0x0000_0000 _0000_0000	
0x3_C800–0x3_C8FF	PKEU parameter memory N	R/W	0x0000_0000 _0000_0000	

1 Depends on the reset configuration word high values. See Section 5.2.4.3.1, “LBLAWBAR0[BASE_ADDR] Reset Value”,” for details.

2 Depends on the reset configuration word high values. See Section 5.2.4.4.1, “LBLAWAR0[EN] and LBLAWAR0[SIZE] Reset Value”,” for details.

3 Depends on the reset configuration word high values. See Section 5.2.4.5.1, “PCILAWBAR0[BASE_ADDR] Reset Value”,” for details.

4 Depends on the reset configuration word high values. See Section 5.2.4.6.1, “PCILAWAR0[EN] and PCILAWAR0[SIZE] Reset Value”,” for details.

5 Depends on the reset configuration word high values. See Section 5.2.4.7.1, “DDRLAWBAR0[BASE_ADDR] Reset Value”,” for details.

6 Depends on the reset configuration word high values. See Section 5.2.4.8.1, “DDRLAWAR0[EN] and DDRLAWAR0[SIZE] Reset Value”,” for details.

7 Depends on the reset configuration word high values.

Chapter 3
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 two lists: one ordered by functional block and one alphabetical.
- List of reset configuration signals
- List of output signal states at reset

NOTE

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

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

3.1 Signals Overview

The signals are grouped as follows:

- DDR memory interface signals
- PCI interface signals
- DUART interface signals
- I^2C interface signals
- Serial peripheral interface signals
- Ethernet management interface signals
- TSEC1 interface signals
- TSEC2 interface signals
- Local bus interface signals
- USB 2.0 port interface signals
- Global timers interface signals
- PIC interface signals
- JTAG, Test, PMC, system control signals
- Clock signals

Figure 3-1 and Figure 3-2 show the external signals of the device and how the signals are grouped. Refer to the *MPC8349EA Integrated Processor Hardware Specifications* 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.

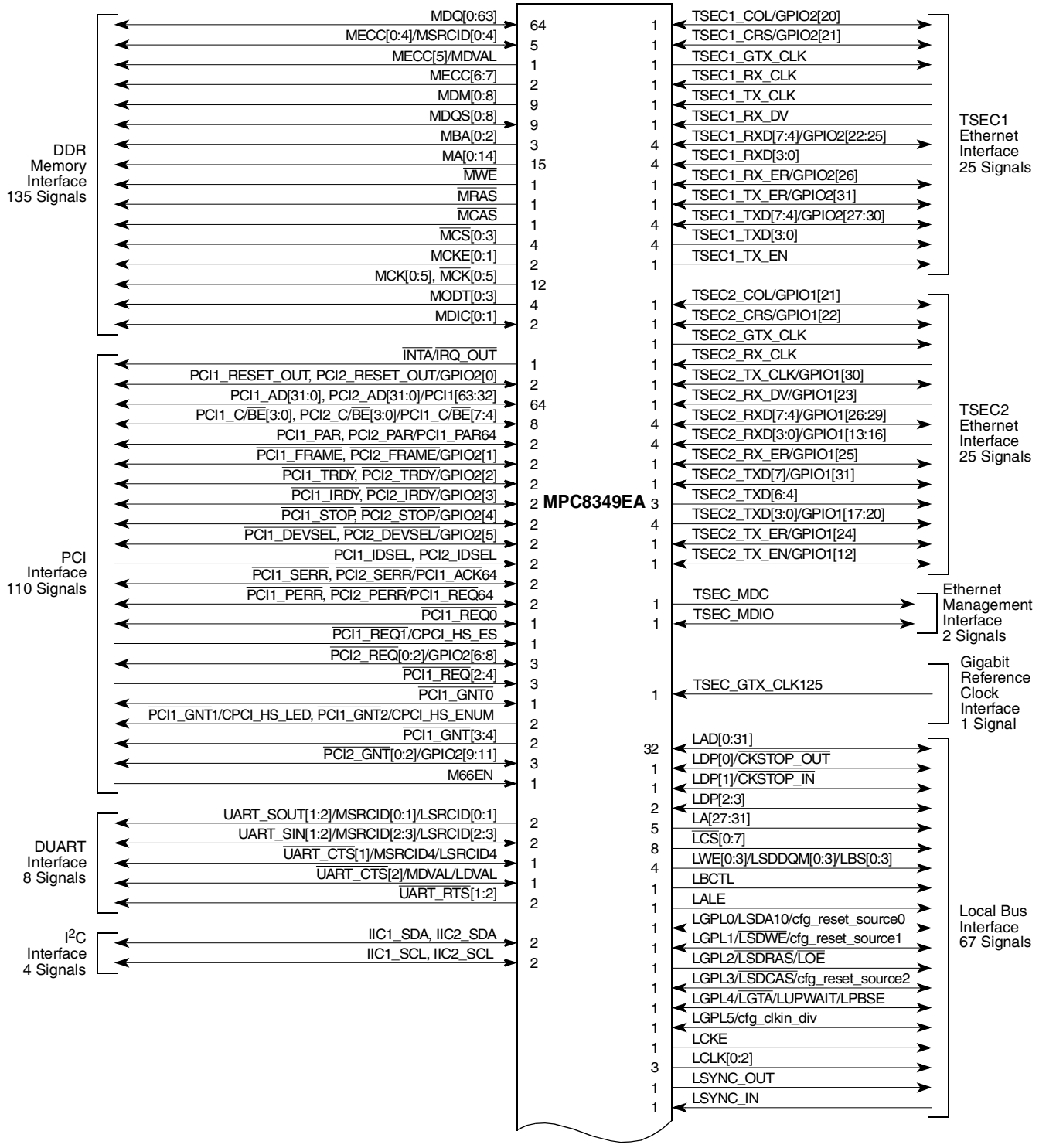


Figure 3-1. MPC8349EA Signal Groupings (1 of 2)

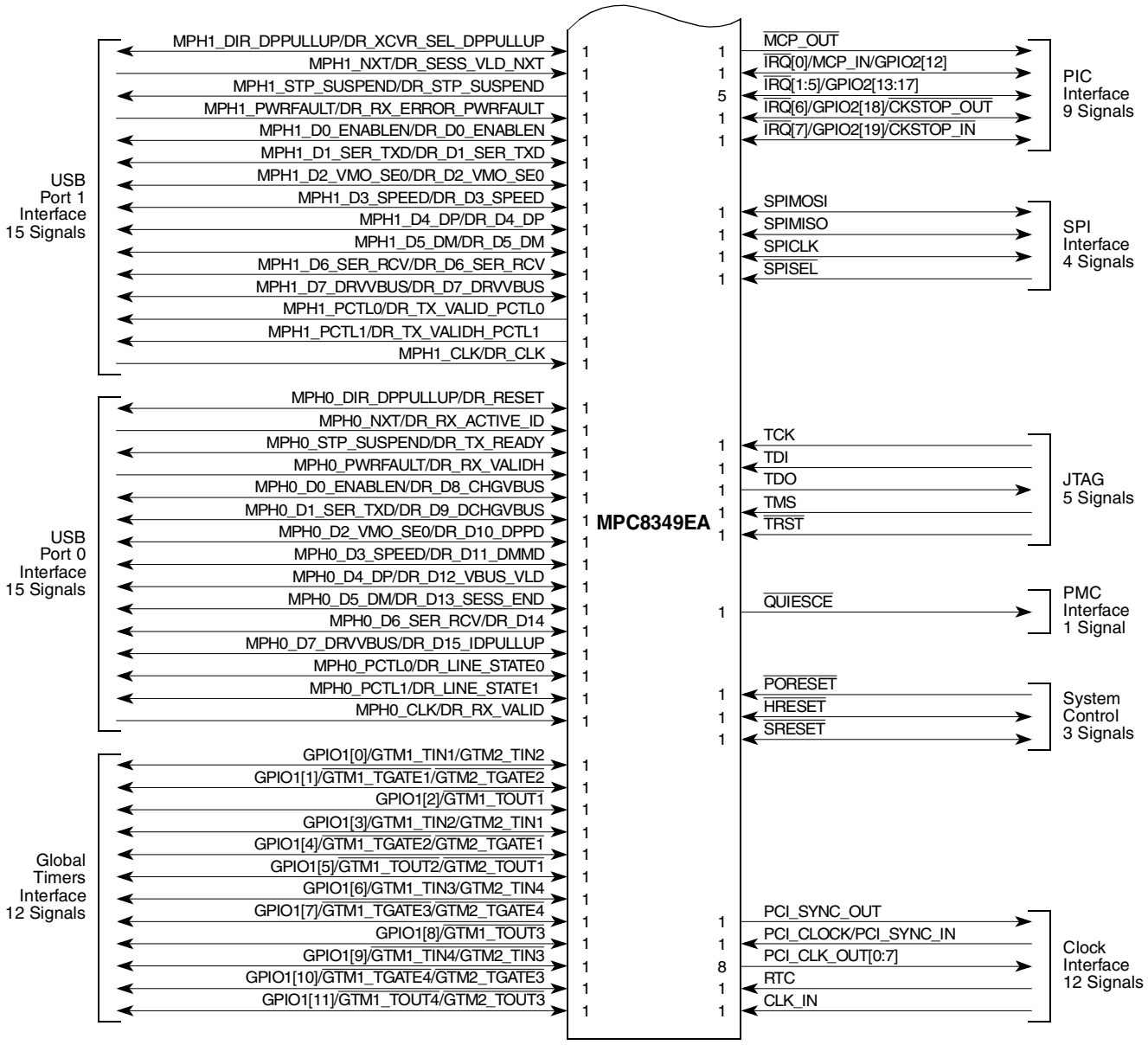


Figure 3-2. MPC8349EA Signal Groupings (2 of 2)

The following tables provide summaries of signal functions. Table 3-1 provides a summary of the signals grouped by function, and Table 3-2 provides 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 the table where the signal function is described.

Table 3-1. MPC8349EA Signal Reference by Functional Block

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
MDQ[0:63]	DDR data	DDR	64	I/O	9-3/9-5	—	—
MECC[0:4]	DDR error correcting code	DDR	5	I/O	9-3/9-5	MSRCID[0:4]	—
MECC[5]	DDR error correcting code	DDR	1	I/O	9-3/9-5	MDVAL	—
MECC[6:7]	DDR error correcting code	DDR	2	I/O	9-3/9-5	—	—
MDM[0:7]	DDR data mask	DDR	8	O	9-3/9-5	—	—
MDM8	DDR ECC data mask	DDR	1	O	9-3/9-5	—	—
MDQS[0:7]	DDR data strobe	DDR	8	I/O	9-3/9-5	—	—
MDQS8	DDR ECC data strobe	DDR	1	I/O	9-3/9-5	—	—
MBA[2:0]	DDR bank select	DDR	3	O	9-3/9-5	—	—
MA[14:0]	DDR address	DDR	15	O	9-3/9-5	—	—
$\overline{\text{MWE}}$	DDR write enable	DDR	1	O	9-3/9-5	—	—
$\overline{\text{MRAS}}$	DDR row address strobe	DDR	1	O	9-3/9-5	—	—
$\overline{\text{MCAS}}$	DDR column address strobe	DDR	1	O	9-3/9-5	—	—
$\overline{\text{MCS}}$[0:3]	DDR chip select (2/DIMM)	DDR	4	O	9-3/9-5	—	—
MCKE[0:1]	DDR clock enable	DDR	2	O	9-4/9-8	—	—
MCK[0:5], $\overline{\text{MCK}}$[0:5]	DDR differential clocks (3 pairs/DIMM)	DDR	12	O	9-4/9-8	—	—
MODT[0:3]	DRAM On-Die Termination	DDR	4	O	9-3/9-5	—	—
MDIC[0:1]	Driver impedance calibration	DDR	2	I/O	9-3/9-5	—	—
$\overline{\text{PCI1_INTA}}$	PCI1 interrupt output	PCI1	1	O	13-3/13-6	$\overline{\text{IRQ_OUT}}$	Table 8-2
$\overline{\text{PCI1_RESET_OUT}}$	PCI1 reset	PCI1	1	O	13-3/13-6	—	—
PCI1_AD[31:0]	PCI1 address/data	PCI1	32	I/O	13-3/13-6	—	—
PCI1_C/$\overline{\text{BE}}$[3:0]	PCI1 command/byte enable	PCI1	4	I/O	13-3/13-6	—	—
PCI1_PAR	PCI1 parity	PCI1	1	I/O	13-3/13-6	—	—
$\overline{\text{PCI1_FRAME}}$	PCI1 frame	PCI1	1	I/O	13-3/13-6	—	—
$\overline{\text{PCI1_TRDY}}$	PCI1 target ready	PCI1	1	I/O	13-3/13-6	—	—
$\overline{\text{PCI1_IRDY}}$	PCI1 initiator ready	PCI1	1	I/O	13-3/13-6	—	—

Table 3-1. MPC8349EA Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
PCI1_STOP	PCI1 stop	PCI1	1	I/O	13-3/13-6	—	—
PCI1_DEVSEL	PCI1 device select	PCI1	1	I/O	13-3/13-6	—	—
PCI1_IDSEL	PCI initial device select	PCI1	1	I	13-3/13-6	—	—
PCI1_SERR	PCI1 system error	PCI1	1	I/O	13-3/13-6	—	—
PCI1_PERR	PCI1 parity error	PCI1	1	I/O	13-3/13-6	—	—
PCI1_REQ[0]	PCI1 request 0	PCI1	1	I/O	13-3/13-6	—	—
PCI1_REQ[1]	PCI1 request 1	PCI1	1	I	13-3/13-6	CPCI_HS_ES	13-3/13-6
PCI1_REQ[2:4]	PCI1 request 2–4	PCI1	3	I	13-3/13-6	—	—
PCI_GNT[0]	PCI1 grant 0	PCI1	1	I/O	13-3/13-6	—	—
PCI_GNT[1]	PCI1 grant 1	PCI1	1	O	13-3/13-6	CPCI_HS_LED	13-3/13-6
PCI_GNT[2]	PCI1 grant 2	PCI1	1	O	13-3/13-6	CPCI_HS_ENUM	13-3/13-6
PCI_GNT[3:4]	PCI1 grant 3–4	PCI1	2	O	13-3/13-6	—	—
PCI2_RESET_OUT	PCI2 reset	PCI2	1	I/O	13-3/13-6	GPIO2[0]	21-1/21-2
PCI2_AD[31:0]	PCI2 address/data	PCI2	32	I/O	13-3/13-6	PCI1[63:32]	13-3/13-6
PCI2_C/BE[3:0]	PCI2 command/ byte enable	PCI2	4	I/O	13-3/13-6	PCI1_C/BE[7:4]	13-3/13-6
PCI2_PAR	PCI2 parity	PCI2	1	I/O	13-3/13-6	PCI1_PAR64	13-3/13-6
PCI2_FRAME	PCI2 frame	PCI2	1	I/O	13-3/13-6	GPIO2[1]	21-1/21-2
PCI2_TRDY	PCI2 target ready	PCI2	1	I/O	13-3/13-6	GPIO2[2]	21-1/21-2
PCI2_IRDY	PCI2 initiator ready	PCI2	1	I/O	13-3/13-6	GPIO2[3]	21-1/21-2
PCI2_STOP	PCI2 stop	PCI2	1	I/O	13-3/13-6	GPIO2[4]	21-1/21-2
PCI2_DEVSEL	PCI2 device select	PCI2	1	I/O	13-3/13-6	GPIO2[5]	21-1/21-2
PCI2_SERR	PCI2 system error	PCI2	1	I/O	13-3/13-6	PCI1_ACK64	13-3/13-6
PCI2_PERR	PCI2 parity error	PCI2	1	I/O	13-3/13-6	PCI1_REQ64	13-3/13-6
PCI2_REQ[0:2]	PCI2 request 0–2	PCI2	3	I/O	13-3/13-6	GPIO2[6:8]	21-1/21-2
PCI2_GNT[0:2]	PCI2 grant 0–2	PCI2	3	I/O	13-3/13-6	GPIO2[9:11]	21-1/21-2
M66EN	PCI 66-MHz timing on/off	PCI1/PCI2	1	I	13-3/13-6	—	—
UART_SOUT[1:2]	DUART serial data out	DUART	2	O	18-2/18-3	MSRCID[0:1]/ LSRCID[0:1]	—/ 10-2/10-5
UART_SIN[1:2]	DUART serial data in	DUART	2	I/O	18-2/18-3	MSRCID[2:3]/ LSRCID[2:3]	—/ 10-2/10-5

Table 3-1. MPC8349EA Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
$\overline{\text{UART_CTS}}$[1]	DUART clear to send	DUART	1	I/O	18-2/18-3	MSRCID4/ LSRCID4	—/ 10-2/10-5
$\overline{\text{UART_CTS}}$[2]	DUART clear to send	DUART	1	I/O	18-2/18-3	MDVAL/LDVAL	—/ 10-2/10-5
$\overline{\text{UART_RTS}}$[1:2]	DUART ready to send	DUART	2	O	18-2/18-3	—	—
IIC1_SDA	I^2C serial data	I^2C1	1	I/O	17-2/17-4	—	—
IIC1_SCL	I^2C serial clock	I^2C1	1	I/O	17-2/17-4	—	—
IIC2_SDA	I^2C serial data	I^2C2	1	I/O	17-2/17-4	—	—
IIC2_SCL	I^2C serial clock	I^2C2	1	I/O	17-2/17-4	—	—
SPIMOSI	SPI master-out slave-in	SPI	1	I/O	19-2/19-7	$\overline{\text{LCS}}$[6]	10-2/10-5
SPIMISO	SPI master-in slave-out	SPI	1	I/O	19-2/19-7	$\overline{\text{LCS}}$[7]	10-2/10-5
SPICLK	SPI clock	SPI	1	I/O	19-2/19-7	—	—
$\overline{\text{SPISEL}}$	SPI slave select	SPI	1	I	19-2/19-7	—	—
TSEC1_COL	TSEC1 collision detect	TSEC1	1	I/O	15-1/15-9	GPIO2[20]	21-1/21-2
TSEC1_CRS	TSEC11 carrier sense	TSEC1	1	I/O	15-1/15-9	GPIO2[21]	21-1/21-2
TSEC1_GTX_CLK	TSEC1 transmit clock out	TSEC1	1	O	15-1/15-9	—	—
TSEC1_RX_CLK	TSEC1 receive clock	TSEC1	1	I	15-1/15-9	—	—
TSEC1_RX_DV	TSEC1 receive data valid	TSEC1	1	I	15-1/15-9	—	—
TSEC1_RXD[7:4]	TSEC1 receive data 7–4	TSEC1	4	I/O	15-1/15-9	GPIO2[22:25]	21-1/21-2
TSEC1_RXD[3:0]	TSEC1 receive data 3–0	TSEC1	4	I	15-1/15-9	—	—
TSEC1_RX_ER	TSEC1 receiver error	TSEC1	1	I/O	15-1/15-9	GPIO2[26]	21-1/21-2
TSEC1_TX_ER	TSEC1 transmit error	TSEC1	1	I/O	15-1/15-9	GPIO2[31]	21-1/21-2
TSEC1_TX_CLK	TSEC1 transmit clock in	TSEC1	1	I	15-1/15-9	—	—
TSEC1_TXD[7:4]	TSEC1 transmit data 7–4	TSEC1	4	I/O	15-1/15-9	GPIO2[27:30]	21-1/21-2
TSEC1_TXD3[1]	TSEC1 transmit data 3	TSEC1	1	I/O	15-1/15-9		—
TSEC1_TXD[2:0]	TSEC1 transmit data 2–0	TSEC1	3	O	15-1/15-9	—	—
TSEC1_TX_EN	TSEC1 transmit enable	TSEC1	1	O	15-1/15-9	—	—

Table 3-1. MPC8349EA Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
TSEC2_COL	TSEC2 collision detect	TSEC2	1	I/O	15-1/15-9	GPIO1[21]	21-1/21-2
TSEC2_CRS	TSEC2 carrier sense	TSEC2	1	I/O	15-1/15-9	GPIO1[22]	21-1/21-2
TSEC2_GTX_CLK	TSEC2 transmit clock out	TSEC2	1	O	15-1/15-9	—	—
TSEC2_RX_CLK	TSEC2 receive clock	TSEC2	1	I	15-1/15-9	—	—
TSEC2_RX_DV	TSEC2 receive data valid	TSEC2	1	I/O	15-1/15-9	GPIO1[23]	21-1/21-2
TSEC2_RXD[7:4]	TSEC2 receive data 7–4	TSEC2	4	I/O	15-1/15-9	GPIO1[26:29]	21-1/21-2
TSEC2_RXD[3:0]	TSEC2 receive data 3–0	TSEC2	4	I/O	15-1/15-9	GPIO1[13:16]	21-1/21-2
TSEC2_RX_ER	TSEC2 receiver error	TSEC2	1	I/O	15-1/15-9	GPIO1[25]	21-1/21-2
TSEC2_TXD[7]	TSEC2 transmit data 7	TSEC2	1	I/O	15-1/15-9	GPIO1[31]	21-1/21-2
TSEC2_TXD[6]	TSEC2 transmit data 6	TSEC2	1	O	15-1/15-9	DR_TERM_SEL	16-4/16-9
TSEC2_TXD[5]	TSEC2 transmit data 5	TSEC2	1	O	15-1/15-9	DR_OPMODE1	16-4/16-9
TSEC2_TXD[4]	TSEC2 transmit data 4	TSEC2	1	O	15-1/15-9	DR_OPMODE0	16-4/16-9
TSEC2_TXD[3:0]	TSEC2 transmit data 3–0	TSEC2	4	I/O	15-1/15-9	GPIO1[17:20]	21-1/21-2
TSEC2_TX_ER	TSEC2 transmit error	TSEC2	1	I/O	15-1/15-9	GPIO1[24]	21-1/21-2
TSEC2_TX_EN	TSEC2 transmit enable	TSEC2	1	I/O	15-1/15-9	GPIO1[12]	21-1/21-2
TSEC2_TX_CLK	TSEC2 transmit clock in	TSEC2	1	I/O	15-1/15-9	GPIO1[30]	21-1/21-2,
TSEC_GTX_CLK125	Gigabit reference clock	TSEC1/ TSEC2	1	I	15-1/15-9	—	—
TSEC_MDC	Ethernet management data clock	Ethernet management	1	I/O	15-1/15-9	—	—
TSEC_MDIO	Ethernet management data in/out	Ethernet management	1	I/O	15-1/15-9	—	—
LAD[0:31]	LBC address/data	LBC	32	I/O	10-2/10-5	—	—
LDP[0]	LBC data parity 0	LBC	1	I/O	10-2/10-5	$\overline{\text{CKSTOP_OUT}}$	4-3/4-4
LDP[1]	LBC data parity 1	LBC	1	I/O	10-2/10-5	$\overline{\text{CKSTOP_IN}}$	4-3/4-4

Table 3-1. MPC8349EA Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
LDP[2:3]	LBC data parity 2–3	LBC	2	I/O	10-2/10-5	$\overline{\text{LCS}}$[4:5]	10-2/10-5
LA[27:31]	LBC port address	LBC	5	O	10-2/10-5	—	—
$\overline{\text{LCS}}$[0:3]	LBC chip select 0–3	LBC	4	O	10-2/10-5	—	—
LWE[0:3]	LBC write enable/ byte lane data mask/ byte select	LBC	4	O	10-2/10-5	LSDDQM[0:3]/ $\overline{\text{LBS}}$[0:3]	3-4/3-20, 10-2/10-5
LBCTL	LBC data buffer control	LBC	1	O	10-2/10-5	—	—
LALE	LBC address latch enable	LBC	1	O	10-2/10-5	—	—
LGPL0/LSDA10	LBC UPM general purpose line 0/SDRAM address bit 10	LBC	1	I/O	10-2/10-5	cfg_reset_source0	4-1/4-1
LGPL1/$\overline{\text{LSDWE}}$	LBC GP line 1/SDRAM write enable	LBC	1	I/O	10-2/10-5	cfg_reset_source1	4-1/4-1
LGPL2/$\overline{\text{LSDRAS}}$/$\overline{\text{LOE}}$	LBC output enable/ SDRAM RAS/GP line 2	LBC	1	O	10-2/10-5	—	—
LGPL3/$\overline{\text{LSDCAS}}$	LBC GP line 3/ SDRAM CAS	LBC	1	I/O	10-2/10-5	cfg_reset_source2	4-1/4-1
LGPL4/$\overline{\text{LGTA}}$/ LUPWAIT/LPBSE	LBC GP line 4/GPCM terminate access/ UPM wait/parity byte select	LBC	1	I/O	10-2/10-5	—	—
LGPL5	LBC GP line 5	LBC	1	I/O	10-2/10-5	cfg_clkin_div	4-1/4-1
LCKE	LBC clock enable	LBC	1	O	10-2/10-5	—	—
LCLK[0:2]	LBC clocks 0–2	LBC	3	O	10-2/10-5	—	—
LSYNC_OUT	LBC DLL synchronization output	LBC	1	O	10-2/10-5	—	—
LSYNC_IN	LBC DLL synchronization input	LBC	1	I	10-2/10-5	—	—
MPH1_D0_ENABLEN	USB host 1 data bit 0	USB Port 1	1	I/O	16-3/16-7	DR_D0_ENABLEN	16-4/16-9
MPH1_D1_SER_TXD	USB host 1 data bit 1	USB Port 1	1	I/O	16-3/16-7	DR_D1_SER_TXD	16-4/16-9
MPH1_D2_VMO_SE0	USB host 1 data bit 2	USB Port 1	1	I/O	16-3/16-7	DR_D2_VMO_SE0	16-4/16-9
MPH1_D3_SPEED	USB host 1 data bit 3	USB Port 1	1	I/O	16-3/16-7	DR_D3_SPEED	16-4/16-9
MPH1_D4_DP	USB host 1 data bit 4	USB Port 1	1	I/O	16-3/16-7	DR_D4_DP	16-4/16-9
MPH1_D5_DM	USB host 1 data bit 5	USB Port 1	1	I/O	16-3/16-7	DR_D5_DM	16-4/16-9

Table 3-1. MPC8349EA Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
MPH1_D6_SER_RCV	USB host 1 data bit 6	USB Port 1	1	I/O	16-3/16-7	DR_D6_SER_RCV	16-4/16-9
MPH1_D7_ DRVVBUS	USB host 1 data bit 7	USB Port 1	1	I/O	16-3/16-7	DR_D7_DRVVBUS	16-4/16-9
MPH1_NXT	USB host 1 next data	USB Port 1	1	I	16-3/16-7	DR_SESS_VLD_NXT	16-4/16-9
MPH1_DIR_ DPPULLUP	USB host 1 data bus direction	USB Port 1	1	I/O	16-3/16-7	DR_XCVR_SEL_ DPPULLUP	16-4/16-9
MPH1_STP_ SUSPEND	USB host 1 data stop/suspend	USB Port 1	1	O	16-3/16-7	DR_STP_SUSPEND	16-4/16-9
MPH1_PWRFAULT	USB host 1 power fault input	USB Port 1	1	I	16-3/16-7	DR_RX_ERROR_ PWRFAULT	16-4/16-9
MPH1_PCTL0	USB host 1 port control 0	USB Port 1	1	O	16-3/16-7	DR_TX_VALID_ PCTL0	16-4/16-9
MPH1_PCTL1	USB host 1 port control 1	USB Port 1	1	O	16-3/16-7	DR_TX_VALIDH_ PCTL1	16-4/16-9
MPH1_CLK	USB host 1 interface clock	USB Port 1	1	I	16-3/16-7	DR_CLK	16-3/16-7
MPH0_D0_ENABLEN	USB host 0 data bit 0	USB Port 0	1	I/O	16-3/16-7	DR_D8_CHGVBUS	16-4/16-9
MPH0_D1_SER_TXD	USB host 0 data bit 1	USB Port 0	1	I/O	16-3/16-7	DR_D9_DCHGVBUS	16-4/16-9
MPH0_D2_VMO_SE0	USB host 0 data bit 2	USB Port 0	1	I/O	16-3/16-7	DR_D10_DPPD	16-4/16-9
MPH0_D3_SPEED	USB host 0 data bit 3	USB Port 0	1	I/O	16-3/16-7	DR_D11_DMPD	16-4/16-9
MPH0_D4_DP	USB host 0 data bit 4	USB Port 0	1	I/O	16-3/16-7	DR_D12_VBUS_VLD	16-4/16-9
MPH0_D5_DM	USB host 0 data bit 5	USB Port 0	1	I/O	16-3/16-7	DR_D13_SESS_END	16-4/16-9
MPH0_D6_SER_RCV	USB host 0 data bit 6	USB Port 0	1	I/O	16-3/16-7	DR_D14	16-4/16-9
MPH0_D7_ DRVVBUS	USB host 0 data bit 7	USB Port 0	1	I/O	16-3/16-7	DR_D15_IDPULLUP	16-4/16-9
MPH0_NXT	USB host 0 next data	USB Port 0	1	I	16-3/16-7	DR_RX_ACTIVE_ID	16-4/16-9
MPH0_DIR_ DPPULLUP	USB host 0 data bus direction	USB Port 0	1	I/O	16-3/16-7	DR_RESET	16-4/16-9
MPH0_STP_ SUSPEND	USB host 0 data stop/suspend	USB Port 0	1	I/O	16-3/16-7	DR_TX_READY	16-4/16-9
MPH0_PWRFAULT	USB host 0 power fault	USB Port 0	1	I	16-3/16-7	DR_RX_VALIDH	16-4/16-9
MPH0_PCTL0	USB host 0 port control 0	USB Port 0	1	I/O	16-3/16-7	DR_LINE_STATE0	16-4/16-9
MPH0_PCTL1	USB host 0 port control 1	USB Port 0	1	I/O	16-3/16-7	DR_LINE_STATE1	16-4/16-9
MPH0_CLK	USB host 0 interface clock	USB Port 0	1	I	16-3/16-7	DR_RX_VALID	16-4/16-9

Table 3-1. MPC8349EA Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
GTM1_TIN1/ GTM2_TIN2	Timer in 1/2	Global Timers	1	I/O	5-53/5-54	GPIO1[0]/ DMA_DREQ0	21-1/21-2 12-1/12-2
GTM1_TGATE1/ GTM2_TGATE2	Timer gate 1/2	Global Timers	1	I/O	5-53/5-54	GPIO1[1]/ DMA_DACK0	21-1/21-2 12-1/12-2
GTM1_TOUT1	Timer out 1	Global Timers	1	I/O	5-53/5-54	GPIO1[2]/ DMA_DDONE0	21-1/21-2 12-1/12-2
GTM1_TIN2/ GTM2_TIN1	Timer in 2/1	Global Timers	1	I/O	5-53/5-54	GPIO1[3]/ DMA_DREQ1	21-1/21-2, 12-1/12-2
GTM1_TGATE2/ GTM2_TGATE1	Timer gate 2/1	Global Timers	1	I/O	5-53/5-54	GPIO1[4]/ DMA_DACK1	21-1/21-2, 12-1/12-2
GTM1_TOUT2/ GTM2_TOUT1	Timer out 2/1	Global Timers	1	I/O	5-53/5-54	GPIO1[5]/ DMA_DDONE1	21-1/21-2, 12-1/12-2
GTM1_TIN3/ GTM2_TIN4	Timer in 3/4	Global Timers	1	I/O	5-53/5-54	GPIO1[6]/ DMA_DREQ2	21-1/21-2 12-1/12-2
GTM1_TGATE3/ GTM2_TGATE4	Timer gate 3/4	Global Timers	1	I/O	5-53/5-54	GPIO1[7]/ DMA_DACK2	21-1/21-2 12-1/12-2
GTM1_TOUT3	Timer out 3	Global Timers	1	I/O	5-53/5-54	GPIO1[8]/ DMA_DDONE2	21-1/21-2 12-1/12-2
GTM1_TIN4/ GTM2_TIN3	Timer in 4/3	Global Timers	1	I/O	5-53/5-54	GPIO1[9]/ DMA_DREQ3	21-1/21-2 12-1/12-2
GTM1_TGATE4/ GTM2_TGATE3	Timer gate 4/3	Global Timers	1	I/O	5-53/5-54	GPIO1[10]/ DMA_DACK3	21-1/21-2 12-1/12-2
GTM1_TOUT4/ GTM2_TOUT3	Timer out 4/3	Global Timers	1	I/O	5-53/5-54	GPIO1[11]/ DMA_DDONE3	21-1/21-2 12-1/12-2
MCP_OUT	Machine check interrupt output	IPIC	1	O	8-2/8-6	—	—
IRQ[0]/MCP_IN	External interrupt 0	IPIC	1	I/O	8-2/8-6	GPIO2[12]	21-1/21-2
IRQ1	External interrupt 1	IPIC	1	I/O	8-2/8-6	GPIO2[13]/ DDR_SR_REQ	21-1/21-2, 9-13/9-22
IRQ[2:5]	External interrupt 2–5	IPIC	4	I/O	8-2/8-6	GPIO2[14:17]	21-1/21-2
IRQ[6]	External interrupt 6	IPIC	1	I/O	8-2/8-6	GPIO2[18]/ CKSTOP_OUT	21-1/21-2, 4-3/4-4
IRQ[7]	External interrupt 7	IPIC	1	I/O	8-2/8-6	GPIO2[19]/ CKSTOP_IN	21-1/21-2, 4-3/4-4
TCK	Test clock	JTAG	1	I	20-2/20-2	—	—
TDI	Test data in	JTAG	1	I	20-2/20-2	—	—
TDO	Test data out	JTAG	1	O	20-2/20-2	—	—
TMS	Test mode select	JTAG	1	I	20-2/20-2	—	—

Table 3-1. MPC8349EA Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
$\overline{\text{TRST}}$	Test reset	JTAG	1	I	20-2/20-2	—	—
TEST_SEL[2]	Test select	TEST	1	I	20-2/20-2	—	—
$\overline{\text{QUIESCE}}$	Quiesce state	PMC	1	O	5-64/5-69	—	—
$\overline{\text{PORESET}}$	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	—	—
PCI_CLK_OUT[0:2]	PCI clock out 0–2	Clocks	3	O	4-2/4-3	—	—
PCI_CLK_OUT[3]	PCI clock out 3	Clocks	1	O	4-2/4-3	$\overline{\text{LCS}}$[6]	10-2/10-5
PCI_CLK_OUT[4]	PCI clock out 4	Clocks	1	O	4-2/4-3	$\overline{\text{LCS}}$[7]	10-2/10-5
PCI_CLK_OUT[5:7]	PCI clock out 5–7	Clocks	3	O	4-2/4-3	—	—
CLK_IN	Clock input	Clocks	1	I	4-2/4-3	—	—
PCI_CLOCK/ PCI_SYNC_IN	PCI clock/PCI clock sync input	Clocks	1	I	4-2/4-3	—	—
PCI_SYNC_OUT	PCI clock sync output	Clocks	1	O	4-2/4-3	—	—
RTC/PIT_CLOCK	Timer clock/ Real time clock	PIT/RTC	1	I	4-2/4-3	—	—

1 This pin should be pulled high during a hard reset for proper functionality of the device since it has a weak internal pull up. No external pull-down resistors are allowed to be mounted on this net.

2 See H/W specification for proper setting of TEST_SEL

Table 3-2 lists the signals in alphabetical order.

Table 3-2. MPC8349EA Alphabetical Signal Reference

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
CLK_IN	Clock input	Clocks	1	I	4-2/4-3	—	—
$\overline{\text{GTM1_TGATE1}}$/ $\overline{\text{GTM2_TGATE2}}$	Timer gate 1/2	Global Timers	1	I/O	5-53/5-54	GPIO1[1]/ $\overline{\text{DMA_DACK0}}$	21-1/21-2 12-1/12-2
$\overline{\text{GTM1_TGATE2}}$/ $\overline{\text{GTM2_TGATE1}}$	Timer gate 2/1	Global Timers	1	I/O	5-53/5-54	GPIO1[4]/ $\overline{\text{DMA_DACK1}}$	21-1/21-2, 12-1/12-2
$\overline{\text{GTM1_TGATE3}}$/ $\overline{\text{GTM2_TGATE4}}$	Timer gate 3/4	Global Timers	1	I/O	5-53/5-54	GPIO1[7]/ $\overline{\text{DMA_DACK2}}$	21-1/21-2 12-1/12-2
$\overline{\text{GTM1_TGATE4}}$/ $\overline{\text{GTM2_TGATE3}}$	Timer gate 4/3	Global Timers	1	I/O	5-53/5-54	GPIO1[10]/ $\overline{\text{DMA_DACK3}}$	21-1/21-2 12-1/12-2

Table 3-2. MPC8349EA Alphabetical Signal Reference (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
GTM1_TIN1/ GTM2_TIN2	Timer in 1/2	Global Timers	1	I/O	5-53/5-54	GPIO1[0]/ DMA_DREQ0	21-1/21-2 12-1/12-2
GTM1_TIN2/ GTM2_TIN1	Timer in 2/1	Global Timers	1	I/O	5-53/5-54	GPIO1[3]/ DMA_DREQ1	21-1/21-2, 12-1/12-2
GTM1_TIN3/ GTM2_TIN4	Timer in 3/4	Global Timers	1	I/O	5-53/5-54	GPIO1[6]/ DMA_DREQ2	21-1/21-2 12-1/12-2
GTM1_TIN4/ GTM2_TIN3	Timer in 4/3	Global Timers	1	I/O	5-53/5-54	GPIO1[9]/ DMA_DREQ3	21-1/21-2 12-1/12-2
$\overline{\text{GTM1_TOUT1}}$	Timer out 1	Global Timers	1	I/O	5-53/5-54	GPIO1[2]/ DMA_DDONE0	21-1/21-2 12-1/12-2
$\overline{\text{GTM1_TOUT2}}$/ $\overline{\text{GTM2_TOUT1}}$	Timer out 2/1	Global Timers	1	I/O	5-53/5-54	GPIO1[5]/ DMA_DDONE1	21-1/21-2, 12-1/12-2
$\overline{\text{GTM1_TOUT3}}$	Timer out 3	Global Timers	1	I/O	5-53/5-54	GPIO1[8]/ DMA_DDONE2	21-1/21-2 12-1/12-2
$\overline{\text{GTM1_TOUT4}}$/ $\overline{\text{GTM2_TOUT3}}$	Timer out 4/3	Global Timers	1	I/O	5-53/5-54	GPIO1[11]/ DMA_DDONE3	21-1/21-2 12-1/12-2
$\overline{\text{HRESET}}$	Hard reset	System control	1	I/O	4-1/4-1	—	—
IIC1_SCL	I^2C serial clock	I^2C1	1	I/O	17-2/17-4	—	—
IIC1_SDA	I^2C serial data	I^2C1	1	I/O	17-2/17-4	—	—
IIC2_SCL	I^2C serial clock	I^2C2	1	I/O	17-2/17-4	—	—
IIC2_SDA	I^2C serial data	I^2C2	1	I/O	17-2/17-4	—	—
$\overline{\text{IRQ}}$[0]/$\overline{\text{MCP_IN}}$	External interrupt 0	IPIC	1	I/O	8-2/8-6	GPIO2[12]	21-1/21-2
$\overline{\text{IRQ}}$[2:5]	External interrupt 2–5	IPIC	4	I/O	8-2/8-6	GPIO2[14:17]	21-1/21-2
$\overline{\text{IRQ}}$[6]	External interrupt 6	IPIC	1	I/O	8-2/8-6	GPIO2[18]/ $\overline{\text{CKSTOP_OUT}}$	21-1/21-2, 4-3/4-4
$\overline{\text{IRQ}}$[7]	External interrupt 7	IPIC	1	I/O	8-2/8-6	GPIO2[19]/ $\overline{\text{CKSTOP_IN}}$	21-1/21-2, 4-3/4-4
$\overline{\text{IRQ1}}$	External interrupt 1	IPIC	1	I/O	8-2/8-6	GPIO2[13]/ DDR_SR_REQ	21-1/21-2, 9-13/9-22
LA[27:31]	LBC port address	LBC	5	O	10-2/10-5	—	—
LAD[0:31]	LBC address/data	LBC	32	I/O	10-2/10-5	—	—
LALE	LBC address latch enable	LBC	1	O	10-2/10-5	—	—
LBCTL	LBC data buffer control	LBC	1	O	10-2/10-5	—	—
LCKE	LBC clock enable	LBC	1	O	10-2/10-5	—	—
LCLK[0:2]	LBC clocks 0–2	LBC	3	O	10-2/10-5	—	—

Table 3-2. MPC8349EA Alphabetical Signal Reference (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
$\overline{\text{LCS}}$[0:3]	LBC chip select 0–3	LBC	4	O	10-2/10-5	—	—
LDP[0]	LBC data parity 0	LBC	1	I/O	10-2/10-5	$\overline{\text{CKSTOP_OUT}}$	4-3/4-4
LDP[1]	LBC data parity 1	LBC	1	I/O	10-2/10-5	$\overline{\text{CKSTOP_IN}}$	4-3/4-4
LDP[2:3]	LBC data parity 2–3	LBC	2	I/O	10-2/10-5	$\overline{\text{LCS}}$[4:5]	10-2/10-5
LGPL0/LSDA10	LBC UPM general purpose line 0/SDRAM address bit 10	LBC	1	I/O	10-2/10-5	cfg_reset_source0	4-1/4-1
LGPL1/$\overline{\text{LSDWE}}$	LBC GP line 1/SDRAM write enable	LBC	1	I/O	10-2/10-5	cfg_reset_source1	4-1/4-1
LGPL2/$\overline{\text{LSDRAS}}$/$\overline{\text{LOE}}$	LBC output enable/ SDRAM RAS/GP line 2	LBC	1	O	10-2/10-5	—	—
LGPL3/$\overline{\text{LSDCAS}}$	LBC GP line 3/ SDRAM CAS	LBC	1	I/O	10-2/10-5	cfg_reset_source2	4-1/4-1
LGPL4/$\overline{\text{LGTA}}$/ LUPWAIT/LPBSE	LBC GP line 4/GPCM terminate access/ UPM wait/parity byte select	LBC	1	I/O	10-2/10-5	—	—
LGPL5	LBC GP line 5	LBC	1	I/O	10-2/10-5	cfg_clkin_div	4-1/4-1
LSYNC_IN	LBC DLL synchronization input	LBC	1	I	10-2/10-5	—	—
LSYNC_OUT	LBC DLL synchronization output	LBC	1	O	10-2/10-5	—	—
LWE[0:3]	LBC write enable/ byte lane data mask/ byte select	LBC	4	O	10-2/10-5	LSDDQM[0:3]/ $\overline{\text{LBS}}$[0:3]	3-4/3-20, 10-2/10-5
M66EN	PCI 66-MHz timing on/off	PCI1/PCI2	1	I	13-3/13-6	—	—
MA[14:0]	DDR address	DDR	15	O	9-3/9-5	—	—
MBA[2:0]	DDR bank select	DDR	3	O	9-3/9-5	—	—
$\overline{\text{MCAS}}$	DDR column address strobe	DDR	1	O	9-3/9-5	—	—
MCK[0:5], $\overline{\text{MCK}}$[0:5]	DDR differential clocks (3 pairs/DIMM)	DDR	12	O	9-4/9-8	—	—
MCKE[0:1]	DDR clock enable	DDR	2	O	9-4/9-8	—	—
$\overline{\text{MCP_OUT}}$	Machine check interrupt output	IPIC	1	O	8-2/8-6	—	—

Table 3-2. MPC8349EA Alphabetical Signal Reference (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
$\overline{\text{MCS}}$[0:3]	DDR chip select (2/DIMM)	DDR	4	O	9-3/9-5	—	—
MDIC[0:1]	Driver impedance calibration	DDR	2	I/O	9-3/9-5	—	—
MDM[0:7]	DDR data mask	DDR	8	O	9-3/9-5	—	—
MDM8	DDR ECC data mask	DDR	1	O	9-3/9-5	—	—
MDQ[0:63]	DDR data	DDR	64	I/O	9-3/9-5	—	—
MDQS[0:7]	DDR data strobe	DDR	8	I/O	9-3/9-5	—	—
MDQS8	DDR ECC data strobe	DDR	1	I/O	9-3/9-5	—	—
MECC[0:4]	DDR error correcting code	DDR	5	I/O	9-3/9-5	MSRCID[0:4]	—
MECC[5]	DDR error correcting code	DDR	1	I/O	9-3/9-5	MDVAL	—
MECC[6:7]	DDR error correcting code	DDR	2	I/O	9-3/9-5	—	—
MODT[0:3]	DRAM On-Die Termination	DDR	4	O	9-3/9-5	—	—
MPH0_CLK	USB host 0 interface clock	USB Port 0	1	I	16-3/16-7	DR_RX_VALID	16-4/16-9
MPH0_D0_ENABLEN	USB host 0 data bit 0	USB Port 0	1	I/O	16-3/16-7	DR_D8_CHGVBUS	16-4/16-9
MPH0_D1_SER_TXD	USB host 0 data bit 1	USB Port 0	1	I/O	16-3/16-7	DR_D9_DCHGVBUS	16-4/16-9
MPH0_D2_VMO_SE0	USB host 0 data bit 2	USB Port 0	1	I/O	16-3/16-7	DR_D10_DPPD	16-4/16-9
MPH0_D3_SPEED	USB host 0 data bit 3	USB Port 0	1	I/O	16-3/16-7	DR_D11_DMPD	16-4/16-9
MPH0_D4_DP	USB host 0 data bit 4	USB Port 0	1	I/O	16-3/16-7	DR_D12_VBUS_VLD	16-4/16-9
MPH0_D5_DM	USB host 0 data bit 5	USB Port 0	1	I/O	16-3/16-7	DR_D13_SESS_END	16-4/16-9
MPH0_D6_SER_RCV	USB host 0 data bit 6	USB Port 0	1	I/O	16-3/16-7	DR_D14	16-4/16-9
MPH0_D7_ DRVVBUS	USB host 0 data bit 7	USB Port 0	1	I/O	16-3/16-7	DR_D15_IDPULLUP	16-4/16-9
MPH0_DIR_ DPPULLUP	USB host 0 data bus direction	USB Port 0	1	I/O	16-3/16-7	DR_RESET	16-4/16-9
MPH0_NXT	USB host 0 next data	USB Port 0	1	I	16-3/16-7	DR_RX_ACTIVE_ID	16-4/16-9
MPH0_PCTL0	USB host 0 port control 0	USB Port 0	1	I/O	16-3/16-7	DR_LINE_STATE0	16-4/16-9
MPH0_PCTL1	USB host 0 port control 1	USB Port 0	1	I/O	16-3/16-7	DR_LINE_STATE1	16-4/16-9
MPH0_PWRFAULT	USB host 0 power fault	USB Port 0	1	I	16-3/16-7	DR_RX_VALIDH	16-4/16-9

Table 3-2. MPC8349EA Alphabetical Signal Reference (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
MPH0_STP_ SUSPEND	USB host 0 data stop/suspend	USB Port 0	1	I/O	16-3/16-7	DR_TX_READY	16-4/16-9
MPH1_CLK	USB host 1 interface clock	USB Port 1	1	I	16-3/16-7	DR_CLK	16-3/16-7
MPH1_D0_ENABLEN	USB host 1 data bit 0	USB Port 1	1	I/O	16-3/16-7	DR_D0_ENABLEN	16-4/16-9
MPH1_D1_SER_TXD	USB host 1 data bit 1	USB Port 1	1	I/O	16-3/16-7	DR_D1_SER_TXD	16-4/16-9
MPH1_D2_VMO_SE0	USB host 1 data bit 2	USB Port 1	1	I/O	16-3/16-7	DR_D2_VMO_SE0	16-4/16-9
MPH1_D3_SPEED	USB host 1 data bit 3	USB Port 1	1	I/O	16-3/16-7	DR_D3_SPEED	16-4/16-9
MPH1_D4_DP	USB host 1 data bit 4	USB Port 1	1	I/O	16-3/16-7	DR_D4_DP	16-4/16-9
MPH1_D5_DM	USB host 1 data bit 5	USB Port 1	1	I/O	16-3/16-7	DR_D5_DM	16-4/16-9
MPH1_D6_SER_RCV	USB host 1 data bit 6	USB Port 1	1	I/O	16-3/16-7	DR_D6_SER_RCV	16-4/16-9
MPH1_D7_ DRVVBUS	USB host 1 data bit 7	USB Port 1	1	I/O	16-3/16-7	DR_D7_DRVVBUS	16-4/16-9
MPH1_DIR_ DPPULLUP	USB host 1 data bus direction	USB Port 1	1	I/O	16-3/16-7	DR_XCVR_SEL_ DPPULLUP	16-4/16-9
MPH1_NXT	USB host 1 next data	USB Port 1	1	I	16-3/16-7	DR_SESS_VLD_NXT	16-4/16-9
MPH1_PCTL0	USB host 1 port control 0	USB Port 1	1	O	16-3/16-7	DR_TX_VALID_ PCTL0	16-4/16-9
MPH1_PCTL1	USB host 1 port control 1	USB Port 1	1	O	16-3/16-7	DR_TX_VALIDH_ PCTL1	16-4/16-9
MPH1_PWRFAULT	USB host 1 power fault input	USB Port 1	1	I	16-3/16-7	DR_RX_ERROR_ PWRFAULT	16-4/16-9
MPH1_STP_ SUSPEND	USB host 1 data stop/suspend	USB Port 1	1	O	16-3/16-7	DR_STP_SUSPEND	16-4/16-9
$\overline{\text{MRAS}}$	DDR row address strobe	DDR	1	O	9-3/9-5	—	—
$\overline{\text{MWE}}$	DDR write enable	DDR	1	O	9-3/9-5	—	—
PCI_CLK_OUT[0:2]	PCI clock out 0–2	Clocks	3	O	4-2/4-3	—	—
PCI_CLK_OUT[3]	PCI clock out 3	Clocks	1	O	4-2/4-3	$\overline{\text{LCS}}$[6]	10-2/10-5
PCI_CLK_OUT[4]	PCI clock out 4	Clocks	1	O	4-2/4-3	$\overline{\text{LCS}}$[7]	10-2/10-5
PCI_CLK_OUT[5:7]	PCI clock out 5–7	Clocks	3	O	4-2/4-3	—	—
PCI_CLOCK/ PCI_SYNC_IN	PCI clock/PCI clock sync input	Clocks	1	I	4-2/4-3	—	—
$\overline{\text{PCI_GNT}}$[0]	PCI1 grant 0	PCI1	1	I/O	13-3/13-6	—	—
$\overline{\text{PCI_GNT}}$[1]	PCI1 grant 1	PCI1	1	O	13-3/13-6	CPCI_HS_LED	13-3/13-6
$\overline{\text{PCI_GNT}}$[2]	PCI1 grant 2	PCI1	1	O	13-3/13-6	CPCI_HS_ENUM	13-3/13-6

Table 3-2. MPC8349EA Alphabetical Signal Reference (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
PCI_GNT[3:4]	PCI1 grant 3–4	PCI1	2	O	13-3/13-6	—	—
PCI_SYNC_OUT	PCI clock sync output	Clocks	1	O	4-2/4-3	—	—
PCI1_AD[31:0]	PCI1 address/data	PCI1	32	I/O	13-3/13-6	—	—
PCI1_C/BE[3:0]	PCI1 command/byte enable	PCI1	4	I/O	13-3/13-6	—	—
PCI1_DEVSEL	PCI1 device select	PCI1	1	I/O	13-3/13-6	—	—
PCI1_FRAME	PCI1 frame	PCI1	1	I/O	13-3/13-6	—	—
PCI1_IDSEL	PCI initial device select	PCI1	1	I	13-3/13-6	—	—
PCI1_INTA	PCI1 interrupt output	PCI1	1	O	13-3/13-6	IRQ_OUT	Table 8-2
PCI1_IRDY	PCI1 initiator ready	PCI1	1	I/O	13-3/13-6	—	—
PCI1_PAR	PCI1 parity	PCI1	1	I/O	13-3/13-6	—	—
PCI1_PERR	PCI1 parity error	PCI1	1	I/O	13-3/13-6	—	—
PCI1_REQ[0]	PCI1 request 0	PCI1	1	I/O	13-3/13-6	—	—
PCI1_REQ[1]	PCI1 request 1	PCI1	1	I	13-3/13-6	CPCI_HS_ES	13-3/13-6
PCI1_REQ[2:4]	PCI1 request 2–4	PCI1	3	I	13-3/13-6	—	—
PCI1_RESET_OUT	PCI1 reset	PCI1	1	O	13-3/13-6	—	—
PCI1_SERR	PCI1 system error	PCI1	1	I/O	13-3/13-6	—	—
PCI1_STOP	PCI1 stop	PCI1	1	I/O	13-3/13-6	—	—
PCI1_TRDY	PCI1 target ready	PCI1	1	I/O	13-3/13-6	—	—
PCI2_AD[31:0]	PCI2 address/data	PCI2	32	I/O	13-3/13-6	PCI1[63:32]	13-3/13-6
PCI2_C/BE[3:0]	PCI2 command/ byte enable	PCI2	4	I/O	13-3/13-6	PCI1_C/BE[7:4]	13-3/13-6
PCI2_DEVSEL	PCI2 device select	PCI2	1	I/O	13-3/13-6	GPIO2[5]	21-1/21-2
PCI2_FRAME	PCI2 frame	PCI2	1	I/O	13-3/13-6	GPIO2[1]	21-1/21-2
PCI2_GNT[0:2]	PCI2 grant 0–2	PCI2	3	I/O	13-3/13-6	GPIO2[9:11]	21-1/21-2
PCI2_IRDY	PCI2 initiator ready	PCI2	1	I/O	13-3/13-6	GPIO2[3]	21-1/21-2
PCI2_PAR	PCI2 parity	PCI2	1	I/O	13-3/13-6	PCI1_PAR64	13-3/13-6
PCI2_PERR	PCI2 parity error	PCI2	1	I/O	13-3/13-6	PCI1_REQ64	13-3/13-6
PCI2_REQ[0:2]	PCI2 request 0–2	PCI2	3	I/O	13-3/13-6	GPIO2[6:8]	21-1/21-2
PCI2_RESET_OUT	PCI2 reset	PCI2	1	I/O	13-3/13-6	GPIO2[0]	21-1/21-2
PCI2_SERR	PCI2 system error	PCI2	1	I/O	13-3/13-6	PCI1_ACK64	13-3/13-6
PCI2_STOP	PCI2 stop	PCI2	1	I/O	13-3/13-6	GPIO2[4]	21-1/21-2

Table 3-2. MPC8349EA Alphabetical Signal Reference (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
PCI2_TRDY	PCI2 target ready	PCI2	1	I/O	13-3/13-6	GPIO2[2]	21-1/21-2
PORESET	Power On reset	System control	1	I	4-1/4-1	—	—
QUIESCE	Quiesce state	PMC	1	O	5-64/5-69	—	—
RTC/PIT_CLOCK	Timer clock/ Real time clock	PIT/RTC	1	I	4-2/4-3	—	—
SPICLK	SPI clock	SPI	1	I/O	19-2/19-7	—	—
SPIMISO	SPI master-in slave-out	SPI	1	I/O	19-2/19-7	LCS[7]	10-2/10-5
SPIMOSI	SPI master-out slave-in	SPI	1	I/O	19-2/19-7	LCS[6]	10-2/10-5
SPISEL	SPI slave select	SPI	1	I	19-2/19-7	—	—
SRESET	Soft reset	System control	1	I/O	4-1/4-1	—	—
TCK	Test clock	JTAG	1	I	20-2/20-2	—	—
TDI	Test data in	JTAG	1	I	20-2/20-2	—	—
TDO	Test data out	JTAG	1	O	20-2/20-2	—	—
TEST_SEL[1]	Test select	TEST	1	I	20-2/20-2	—	—
TMS	Test mode select	JTAG	1	I	20-2/20-2	—	—
TRST	Test reset	JTAG	1	I	20-2/20-2	—	—
TSEC_GTX_CLK125	Gigabit reference clock	TSEC1/ TSEC2	1	I	15-1/15-9	—	—
TSEC_MDC	Ethernet management data clock	Ethernet management	1	I/O	15-1/15-9	—	—
TSEC_MDIO	Ethernet management data in/out	Ethernet management	1	I/O	15-1/15-9	—	—
TSEC1_COL	TSEC1 collision detect	TSEC1	1	I/O	15-1/15-9	GPIO2[20]	21-1/21-2
TSEC1_CRS	TSEC11 carrier sense	TSEC1	1	I/O	15-1/15-9	GPIO2[21]	21-1/21-2
TSEC1_GTX_CLK	TSEC1 transmit clock out	TSEC1	1	O	15-1/15-9	—	—
TSEC1_RX_CLK	TSEC1 receive clock	TSEC1	1	I	15-1/15-9	—	—
TSEC1_RX_DV	TSEC1 receive data valid	TSEC1	1	I	15-1/15-9	—	—
TSEC1_RX_ER	TSEC1 receiver error	TSEC1	1	I/O	15-1/15-9	GPIO2[26]	21-1/21-2
TSEC1_RXD[3:0]	TSEC1 receive data 3–0	TSEC1	4	I	15-1/15-9	—	—

Table 3-2. MPC8349EA Alphabetical Signal Reference (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
TSEC1_RXD[7:4]	TSEC1 receive data 7–4	TSEC1	4	I/O	15-1/15-9	GPIO2[22:25]	21-1/21-2
TSEC1_TX_CLK	TSEC1 transmit clock in	TSEC1	1	I	15-1/15-9	—	—
TSEC1_TX_EN	TSEC1 transmit enable	TSEC1	1	O	15-1/15-9	—	—
TSEC1_TX_ER	TSEC1 transmit error	TSEC1	1	I/O	15-1/15-9	GPIO2[31]	21-1/21-2
TSEC1_TXD[2:0]	TSEC1 transmit data 2–0	TSEC1	3	O	15-1/15-9	—	—
TSEC1_TXD[7:4]	TSEC1 transmit data 7–4	TSEC1	4	I/O	15-1/15-9	GPIO2[27:30]	21-1/21-2
TSEC1_TXD3	TSEC1 transmit data 3	TSEC1	1	I/O	15-1/15-9		—
TSEC2_COL	TSEC2 collision detect	TSEC2	1	I/O	15-1/15-9	GPIO1[21]	21-1/21-2
TSEC2_CRS	TSEC2 carrier sense	TSEC2	1	I/O	15-1/15-9	GPIO1[22]	21-1/21-2
TSEC2_GTX_CLK	TSEC2 transmit clock out	TSEC2	1	O	15-1/15-9	—	—
TSEC2_RX_CLK	TSEC2 receive clock	TSEC2	1	I	15-1/15-9	—	—
TSEC2_RX_DV	TSEC2 receive data valid	TSEC2	1	I/O	15-1/15-9	GPIO1[23]	21-1/21-2
TSEC2_RX_ER	TSEC2 receiver error	TSEC2	1	I/O	15-1/15-9	GPIO1[25]	21-1/21-2
TSEC2_RXD[3:0]	TSEC2 receive data 3–0	TSEC2	4	I/O	15-1/15-9	GPIO1[13:16]	21-1/21-2
TSEC2_RXD[7:4]	TSEC2 receive data 7–4	TSEC2	4	I/O	15-1/15-9	GPIO1[26:29]	21-1/21-2
TSEC2_TX_CLK	TSEC2 transmit clock in	TSEC2	1	I/O	15-1/15-9	GPIO1[30]	21-1/21-2,
TSEC2_TX_EN	TSEC2 transmit enable	TSEC2	1	I/O	15-1/15-9	GPIO1[12]	21-1/21-2
TSEC2_TX_ER	TSEC2 transmit error	TSEC2	1	I/O	15-1/15-9	GPIO1[24]	21-1/21-2
TSEC2_TXD[3:0]	TSEC2 transmit data 3–0	TSEC2	4	I/O	15-1/15-9	GPIO1[17:20]	21-1/21-2
TSEC2_TXD[4]	TSEC2 transmit data 4	TSEC2	1	O	15-1/15-9	DR_OPMODE0	16-4/16-9
TSEC2_TXD[5]	TSEC2 transmit data 5	TSEC2	1	O	15-1/15-9	DR_OPMODE1	16-4/16-9
TSEC2_TXD[6]	TSEC2 transmit data 6	TSEC2	1	O	15-1/15-9	DR_TERM_SEL	16-4/16-9

Table 3-2. MPC8349EA Alphabetical Signal Reference (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
TSEC2_TXD[7]	TSEC2 transmit data 7	TSEC2	1	I/O	15-1/15-9	GPIO1[31]	21-1/21-2
$\overline{\text{UART_CTS}}$[1]	DUART clear to send	DUART	1	I/O	18-2/18-3	MSRCID4/ LSRCID4	—/ 10-2/10-5
$\overline{\text{UART_CTS}}$[2]	DUART clear to send	DUART	1	I/O	18-2/18-3	MDVAL/LDVAL	—/ 10-2/10-5
$\overline{\text{UART_RTS}}$[1:2]	DUART ready to send	DUART	2	O	18-2/18-3	—	—
UART_SIN[1:2]	DUART serial data in	DUART	2	I/O	18-2/18-3	MSRCID[2:3]/ LSRCID[2:3]	—/ 10-2/10-5
UART_SOUT[1:2]	DUART serial data out	DUART	2	O	18-2/18-3	MSRCID[0:1]/ LSRCID[0:1]	—/ 10-2/10-5

1 See H/W specification for proper setting of TEST_SEL

3.2 Configuration Signals Sampled at Reset

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

Table 3-3. Reset Configuration Signals

Functional Interface	Functional Signal Name	Reset Configuration Name
LBC	LSDA10/LGPL0	cfg_reset_source0
	$\overline{\text{LSDWE}}$/LGPL1	cfg_reset_source1
	$\overline{\text{LSDCAS}}$/LGPL3	cfg_reset_source2
	LGPL5	cfg_clkin_div

3.3 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 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. An exception is the state of the signal PCI2_RESET_OUT, as shown in Table 3-5.

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 3-4 shows the states of the output-only signals.

Table 3-4. Output Signal States During System Reset

Interface	Signal	State During Reset
MDM[0:8]	DDR data mask	All ‘0’
MBA[0:2]	DDR bank select	All ‘Z’
MA[0:14]	DDR address	All ‘Z’
$\overline{\text{MWE}}$	DDR write enable	‘Z’
$\overline{\text{MRAS}}$	DDR row address strobe	‘Z
$\overline{\text{MCAS}}$	DDR column address strobe	‘Z’
$\overline{\text{MCS}}$[0:3]	DDR chip select (2/DIMM)	All ‘Z’
MCKE[0:1]	DDR clock enable	All ‘0’
MCK[0:5]	DDR differential clocks	All ‘0’
$\overline{\text{MCK}}$[0:5]	DDR differential clocks	All ‘0’
MODT[0:3]	DRAM On-Die Termination	All ‘0’
$\overline{\text{INTA}}$/ $\overline{\text{IRQ_OUT}}$	PCI1 interrupt output	‘Z”
$\overline{\text{PCI1_RESET_OUT}}$	PCI1 reset output	‘0’
$\overline{\text{PCI1_GNT}}$[1:4]	PCI1 grant 1–4	All ‘Z’
UART_SOUT[1:2]	DUART serial data out	All ‘Z’
$\overline{\text{UART_RTS}}$[1:2]	DUART ready to send	11’
TSEC1_GTX_CLK	TSEC1 transmit clock out	‘0’
TSEC1_TXD[2:0]	TSEC1 transmit data 2–0	All ‘0’
TSEC1_TX_EN	TSEC1 transmit enable	‘0’
TSEC2_GTX_CLK	TSEC2 transmit clock out	‘0’
TSEC2_TXD[6:4]	TSEC2 transmit data 6-4	All ‘Z’
TSEC_MDC	Ethernet management data clock	‘0’
LA[27:31]	LBC port address	Active—used to load reset configuration word
$\overline{\text{LCS}}$[0]	LBC chip select 0	Active—used to load reset configuration word
$\overline{\text{LCS}}$[1:3]	LBC chip select 1–3	All ‘1’
$\overline{\text{LWE}}$[0:3]/ LSDDQM[0:3]/ $\overline{\text{LBS}}$[0:3]	LBC write enable/byte lane data mask/byte select	All ‘1’
LBCTL	LBC data buffer control	Active—used to load reset configuration word
LALE	LBC address latch enable	Active—used to load reset configuration word

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

Interface	Signal	State During Reset
$\overline{\text{LOE}}$/$\overline{\text{LSDRAS}}$/LGPL2	LBC output enable/SDRAM RAS/GP line 2	Active—used to load reset configuration word
LCKE	LBC clock enable	'1'
LCLK[0:2]	LBC clock	All '0'
LSYNC_OUT	LBC clock synchronization output	'0'
$\overline{\text{DMA_DACK}}$[0:3]	DMA acknowledge 0–3	
$\overline{\text{DMA_DDONE}}$[0:3]	DMA done 0–3	
MPH1_STP_SUSPEND	USB host 1 stop data suspend	'Z'
MPH1_PCTL0	USB host 1 port control 0	'Z'
MPH1_PCTL1	USB host 1 port control 1	'Z'
$\overline{\text{MCP_OUT}}$	Machine check interrupt output	'Z'
TDO	Test data out	'Z'
$\overline{\text{QUIESCE}}$	Quiesce state	'1'
PCI_CLK_OUT[0:7]	PCI clock 0–7	All '0'
PCI_SYNC_OUT	PCI sync output	Active clock

Table 3-5 provides the list of bidirectional signals that are not disabled during reset. Note that because PCI2_RESET_OUT is active during reset, the usage of GPIO2[0] (in cases when PCI2 is not needed) is restricted to output only.

Table 3-5. Bidirectional Signals Active During Reset

Interface	Signal	State During Reset
$\overline{\text{PCI2_RESET_OUT}}$	PCI2 reset output	'0'

Table 3-6 provides the list of registers that control the device's signal multiplexing.

Table 3-6. Signals for Multiplexing

Signal Group	Multiplexing is Controlled By	Table/Page
PCI2	RCWH[PCI64]	4.3.2.2/4-14
PCI1/CPCI	RCWH[PCI1ARB]	4.3.2.2/4-14
All others	SICRL/SICRH	5.3.2.5/5-21/5.3.2.6/5-24

Chapter 4
Reset, Clocking, and Initialization

The reset, clocking, and control signals offer many options for the 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.	
		State Meaning	Asserted—An external agent has triggered a power-on reset sequence. Negated—No power-on reset.
		Timing	See the hardware specifications for timing information.
		Reset State	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. During $\overline{\text{HRESET}}$, $\overline{\text{SRESET}}$ is asserted. $\overline{\text{HRESET}}$ is an open-drain signal.	
		State Meaning	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.
		Timing	Assertion—Occur at any time, asynchronously to any clock. Negation—Must be asserted for at least 32 CLKIN (PCI host mode) or PCI_CLK (PCI agent mode) cycles.
		Requirements	An open-drain signal. An external pull-up is required.
		Reset State	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	
SRESET	I/O	Soft reset. Causes the device to abort all current internal transactions, set most registers to their default values, and cause the core to enter its reset state. The I/O signal functionality and direction as well as the memory controller operation are unaffected by SRESET. SRESET can be asserted completely asynchronously with respect to all other signals. The device can detect an external assertion of SRESET while the device is not asserting hard or soft reset. SRESET is an open-drain signal.	
		State Meaning	Asserted—An external agent or internal hardware has triggered a soft reset sequence. The internal hardware drives SRESET until the sequence completes.
		Timing	Assertion—Occurs at any time, asynchronously to any clock. Negation—Must be asserted for at least 32 CLKIN (PCI host mode) or PCI_CLK (PCI agent mode) cycles.
		Requirements	An open-drain signal. An external pull-up is required.
		Reset State	Output, driven low during power-on, hard reset, and soft reset flows. High impedance after reset flow completes.
CFG_RESET_SOURCE[0:2]	I	Reset configuration word source selection. These signals are located on device pins that have other functions when the device is not in reset. These input signals are sampled during the assertion of PORESET to determine the interface from which the device loads the reset configuration words.	
		State Meaning	See Section 4.3.1.1, “Reset Configuration Word Source.”
		Timing	These input signals are sampled during the assertion of PORESET after a stable clock is supplied (PORESET flow) and must be pulled high or low by external resistors as long as HRESET is asserted.
		Requirements	During PORESET and HRESET flows, all other signal drivers connected to these signals must be in the high-impedance state. Refer to the hardware specifications for proper resistor values to pull reset configuration signals high or low.
		Reset State	Input during power-on and hard reset flows. Functional signal after reset flow completes.
CFG_CLKIN_DIV	I	Clock in division selection. This signal is located on a device pin that has another function when the device is not in reset state. This signal is sampled during the assertion of PORESET to determine whether CLKIN is divided by two.	
		State Meaning	See Section 4.3.1.2, “CLKIN Division.”
		Timing	This signal is sampled during the assertion of PORESET after a stable clock is supplied (PORESET flow), and it must be pulled high or low by external resistors as long as HRESET is asserted.
		Requirements	During PORESET and HRESET flows, all other signal drivers connected to this signal must be in the high-impedance state. Refer to the hardware specifications for proper resistors values to pull reset configuration signals high or low.
		Reset State	Input during power-on and hard reset flows. Functional signal after reset flow completes.

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 4-9 for the internal distribution of clocks in the device.

Table 4-2. External Clock Signals

Signal	I/O	Description	
CLKIN	I	System clock. In PCI host mode, CLKIN is the primary input clock. CLKIN directly feeds the PCI output clock dividers and is driven out on the PCI_SYNC_OUT signal for de-skewing external PCI clocks routing. In PCI agent mode, this signal should be tied to GND.	
		Timing	Assertion/Negation—See the hardware specifications for timing information.
		Requirements	Should be tied low in PCI agent mode.
		Reset State	Always input.
PCI_CLK/ PCI_SYNC_IN	I	PCI clock/ PCI synchronization clock (PCI_CLK/PCI_SYNC_IN). In PCI agent mode, PCI_CLK is the primary clock input to the device. In PCI host mode, PCI_SYNC_IN is connected externally to PCI_SYNC_OUT.	
		Timing	Assertion/Negation—See the hardware specifications for timing information
		Reset State	Always input.
PCI_SYNC_OUT	O	Reference PCI output synchronization clock (PCI_SYNC_OUT). In PCI host mode, PCI_SYNC_OUT is connected externally to PCI_SYNC_IN signal for de-skewing external PCI clocks routing. PCI_SYNC_OUT has the same frequency as CLKIN or CLKIN/2 depending on the state of CFG_CLKIN_DIV at reset. See Section 4.3.1.2, "CLKIN Division." In PCI agent mode, this signal is typically not used.	
		Timing	Assertion/Negation—See the hardware specifications for timing information.
		Reset State	Always output, toggling in PCI host mode.
PCI_CLK_OUT[0:7]	O	PCI output clocks bank. In PCI host mode, the device provides eight separate clock output signals for feeding PCI agent devices.	
		Timing	Assertion/Negation—See the hardware specifications for timing information.
		Reset State	Always output. Drive '0' during and after power-on reset flow. Enabled by a memory-mapped register.

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 several inputs to the reset logic:

- Power-on reset ($\overline{\text{PORESET}}$)
- External hard reset ($\overline{\text{HRESET}}$)
- External soft reset ($\overline{\text{SRESET}}$)

- Software watchdog reset
- System bus monitor reset
- Checkstop reset
- JTAG 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 sources 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 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. During $\overline{\text{HRESET}}$, $\overline{\text{SRESET}}$ is asserted. $\overline{\text{HRESET}}$ is an open-drain signal.
Soft reset ($\overline{\text{SRESET}}$)	Bidirectional I/O signal. The device can detect an external assertion of $\overline{\text{SRESET}}$ only while it is not asserting hard or soft reset. $\overline{\text{SRESET}}$ is an open-drain 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.
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.
JTAG reset	When JTAG logic asserts the JTAG soft reset signal, an internal soft reset sequence is generated.
Software hard reset	A hard reset sequence can be initialized by writing to a memory-mapped register (RCR)
Software soft reset	A soft 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.
- Hard reset resets the entire device excluding clock logic and error capture registers
- Soft reset initializes the internal logic while maintaining the system configuration.

All reset types generate a reset to the core, and the impact on the application is that the core resets the MSR[IP] to the value in the BMS field of the reset configuration word high, see Section 4.3.2.2, "Reset Configuration Word High Register (RCWHR)."

The memory controller, system protection logic, interrupt controller, and I/O signals are initialized only on hard reset. Soft reset initializes the internal logic while maintaining the system configuration. Asserting external SRESET generates a hard reset to the core and to the remainder of the device. 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**	**JTAG Reset External Soft Reset**
Resets: PLLs, clocks, RTC unit, and error capture registers	**Yes**	No	No
Resets: DDR, LBC, I/O multiplexors, GTM, PIT, GPIO, system configuration, and local access windows	**Yes**	Yes	No
Resets other internal logic	**Yes**	Yes	Yes
Reset configuration words loaded	**Yes**	Yes	No
HRESET driven	**Yes**	Yes	No
SRESET driven	**Yes**	Yes	Yes
Hard reset to e300 core	**Yes**	Yes	Yes

4.2.2 Power-On Reset Flow

Assertion of the PORESET external signal initiates the power-on reset flow. 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 PORESET, the device starts the configuration process. The device asserts HRESET and SRESET throughout the power-on reset process, including configuration. Configuration time varies according to the configuration source and CLKIN (PCI host mode) or PCI_CLK (PCI agent mode) frequency. Initially, the reset configuration inputs are sampled to determine the configuration source and the input clock division mode. 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, HRESET is released; SRESET is released 16 clocks later.

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 (some clock, clock enabled, and system control signals remain active).
3. The system applies a stable CLKIN (PCI host mode) or PCI_CLK (PCI agent mode) signal and stable reset configuration inputs (CFG_RESET_SOURCE, CFG_CLKIN_DIV).
4. The system negates $\overline{\text{PORESET}}$ after at least 32 stable CLKIN (PCI host mode) or PCI_CLK (PCI agent mode) clock cycles.
5. The device samples the reset configuration input signals to determine the clock division and 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.

NOTE:

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 $\overline{\text{PORESET}}$. There is no need to assert the $\overline{\text{SRESET}}$ signal when $\overline{\text{HRESET}}$ is asserted.

11. The internal reset to the core and the rest of the logic is negated. I/O drivers are enabled. The LBC DLL begins to lock. The PCI interfaces can assert $\overline{\text{DEVSEL}}$ in response to configuration cycles.
12. The device stops driving $\overline{\text{SRESET}}$ and $\overline{\text{SRESET}}$ is negated. 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. Before the boot sequencer finishes, it can enable the PCI interfaces to accept external requests, if required, by clearing CFG_LOCK bit in the PCI bus function register as described in Table 13-41. 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).”
14. The PCI interface can now accept external requests, if enabled, and 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.

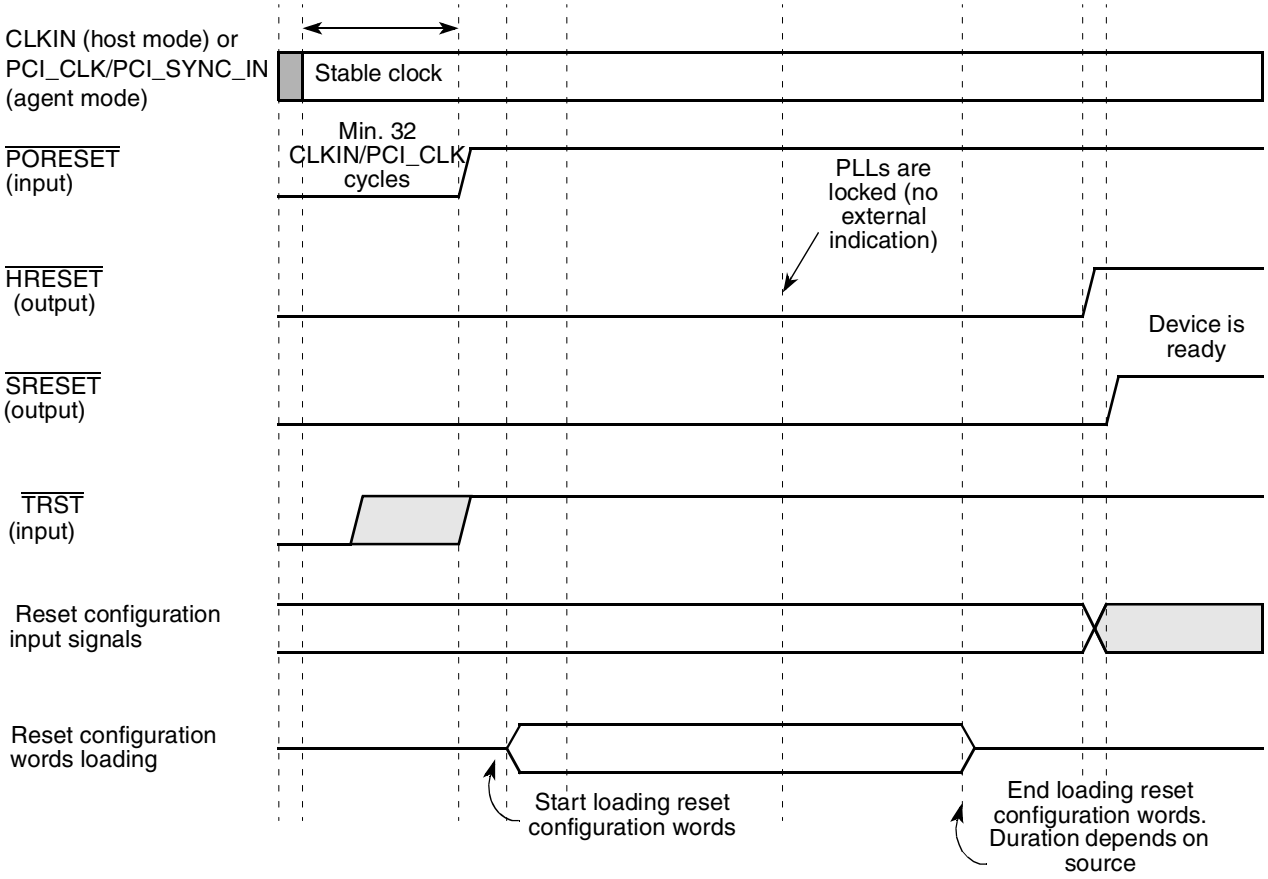


Figure 4-1. Power-On Reset Flow

4.2.3 Hard Reset Flow

The HRESET signal is initiated externally by asserting HRESET or internally when the device detects a reason to generate an internal hard reset sequence. In both cases the device continues asserting HRESET and SRESET throughout the HRESET state. The hard reset sequence time varies according to the configuration source and CLKIN (PCI host mode) or PCI_CLK (PCI agent mode) frequency. The reset configuration input signals (CFG_RESET_SOURCE and CFG_CLKIN_DIV) are 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 both the HRESET and SRESET signals and exits the 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/soft) reset.

NOTE

Because the device does not sample the reset configuration input signals (CFG_RESET_SOURCE, CFG_CLKIN_DIV) during a hard reset flow, setting a new value on those signals (other than what was set during power-on reset) has no effect.

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

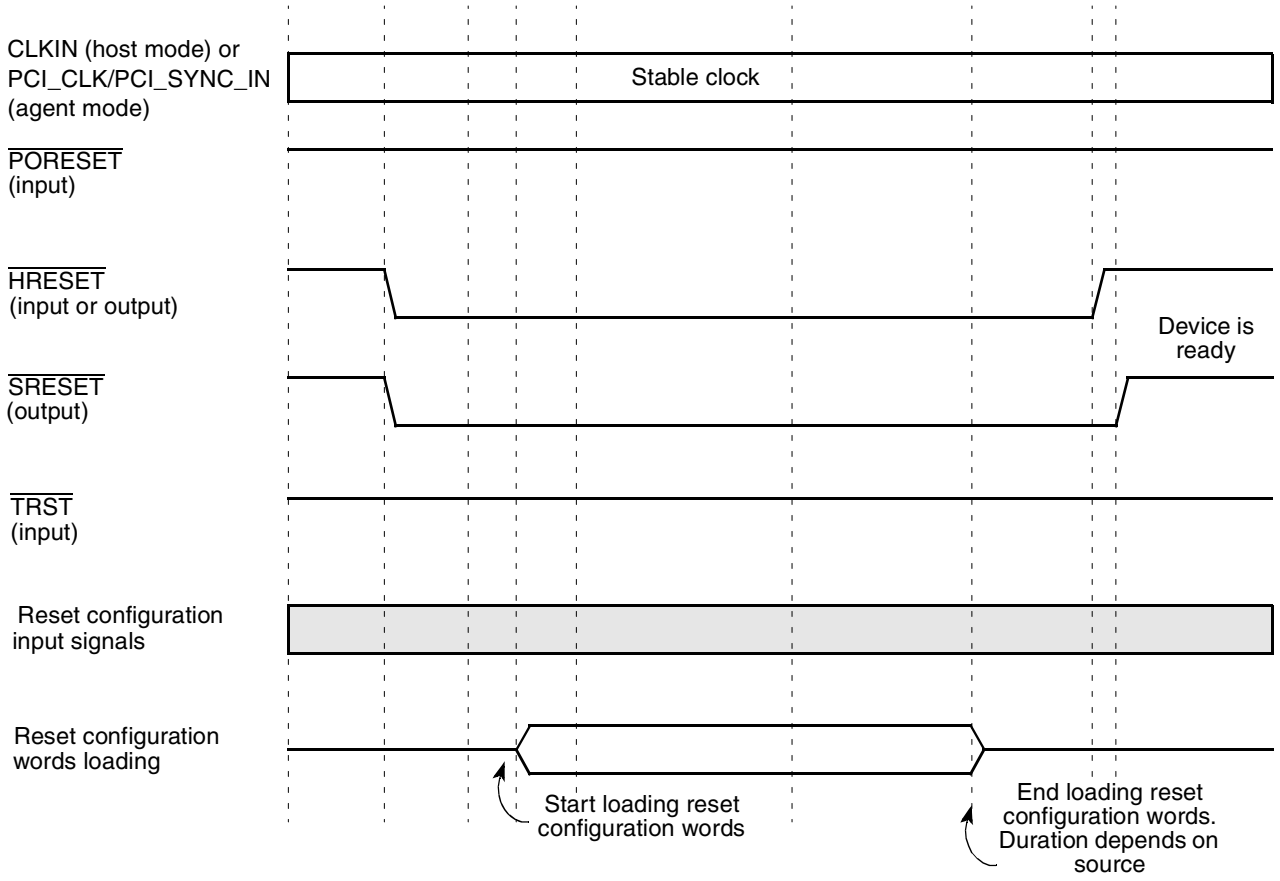


Figure 4-2. Hard Reset Flow

4.2.4 Soft Reset Flow

The SRESET signal can be initiated externally by asserting SRESET or internally when the device detects a cause to assert SRESET. In both cases the device asserts SRESET for 512 PCI_CLK/PCI_SYNC_IN/ SYNC_IN clock cycles, after which the device releases SRESET and exits the SRESET signal. An external pull-up resistor should negate SRESET; after negation is detected, a 16-cycle period is taken before testing for the presence of an external (hard/soft) reset. While SRESET is asserted, internal hardware is reset but hard reset configuration does not change.

4.3 Reset Configuration

The device is initialized using two complementary methods, latching CFG_RESET_SOURCE and loading the reset configuration words. Initially, a few input signals are 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 located 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 $\overline{\text{PORESET}}$, after a stable clock is supplied ($\overline{\text{PORESET}}$), and must be pulled high or low by external resistors as long as $\overline{\text{HRESET}}$ is asserted. During the period in which the $\overline{\text{PORESET}}$ and $\overline{\text{HRESET}}$ signals are asserted, all other signal drivers connected to these signals must be in the high-impedance state. Refer to the hardware specifications for proper resistor values for pulling reset configuration signals high or low.

This section describes the modes configured by the reset configuration signals. Note that the reset configuration inputs sampled values are accessible to software through memory-mapped registers 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 pullup or pulldown 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 a local bus EEPROM, or I^2C EEPROM (I^2C #1) 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[0:2]	Meaning
000	Reset configuration word is loaded from a local bus EEPROM.
001	Reset configuration word is loaded from an I^2C EEPROM. PCI_CLK/PCI_SYNC_IN is in the range of 25–44 MHz. This option will be removed from future designs. Thus, customers are recommended to use 010 option.
010	Reset configuration word is loaded from an I^2C EEPROM. PCI_CLK/PCI_SYNC_IN is valid for any PCI frequency up to 66.666 MHz (range of 24–66.666 MHz).
011	Hard-coded option 0. Reset configuration word is not loaded.
100	Hard-coded option 1. Reset configuration word is not loaded.
101	Hard-coded option 2. Reset configuration word is not loaded.
110	Hard-coded option 3. Reset configuration word is not loaded.
111	Hard-coded option 4. Reset configuration word is not loaded.

4.3.1.2 CLKIN Division

When the device is configured as a PCI host, the CFG_CLKIN_DIV configuration input selects the relationship between CLKIN and PCI_SYNC_OUT/SYNC_OUT as shown in Table 4-6. As a PCI host, the device supports eight PCI_CLK output signals. The frequency of each output clock can be programmed in the OCCR register to be equal to or half of the CLKIN frequency. The CFG_CLKIN_DIV reset configuration signal should be pulled high if there is at least one output PCI clock with half the frequency of CLKIN and pulled low otherwise.

When the device is configured as a PCI agent, the CFG_CLKIN_DIV configuration input can be used to double the internal clock frequencies, if sampled as '1' during power-on reset assertion. This feature is useful if a fixed internal frequency is desired regardless of whether the PCI clock is running at 33 or 66 MHz. PCI specifications require the PCI clock frequency information to be provided by the M66EN signal.

Table 4-6. CLKIN Division

CFG_CLKIN_DIV	Description
0	In PCI host mode, CLKIN: PCI_SYNC_OUT = 1:1 and all PCI_CLK_OUT[0:7] clocks are running at a frequency which is equal to the CLKIN frequency
1	In PCI host mode, CLKIN: PCI_SYNC_OUT = 2:1 and the PCI_CLK_OUT[0:7] clocks can be programmed to CLKIN/2 in the OCCR. In PCI agent mode, internal frequency is doubled. Refer to the *MPC8349EA Hardware Specifications* for details.

4.3.1.3 Selecting Reset Configuration Input Signals

The example described in Table 4-7 shows how the user should pull down or pull up the reset configuration input signals (CFG_RESET_SOURCE, CFG_CLKIN_DIV). The reset sequence duration is measured from the negation of $\overline{\text{PORESET}}$ to the negation of $\overline{\text{SRESET}}$. Note that the duration mentioned in this table is typical and doesn't represent cases in which the process of loading the reset configuration word had to be retried due to errors.

Table 4-7. Selecting Reset Configuration Input Signals

I^2C EEPROM Configuration Words	CLKIN Frequency (Host Mode)	CFG_CLKIN_DIV (Host Mode)	PCI_CLK Frequency (Agent Mode)	CFG_RESET_SOURCE[0:2]	Reset Sequence Duration in CLKIN/PCI_CLK Cycles	Duration
No	33 MHz	0	33 MHz	000, 011–111 (not I^2C EEPROM)	15380	462 µs
No	66 MHz	0	66 MHz	000, 011–111 (not I^2C EEPROM)	15380	231 µs
No	66 MHz	1	33 MHz	000, 011–111 (not I^2C EEPROM)	30760/15380	462 µs
Yes	33 MHz	0	33 MHz	001 (I^2C EEPROM, low PCI_SYNC_IN/ PCI_CLK clock frequency)	24548	736 µs

Table 4-7. Selecting Reset Configuration Input Signals (continued)

I^2C EEPROM Configuration Words	CLKIN Frequency (Host Mode)	CFG_CLKIN_DIV (Host Mode)	PCI_CLK Frequency (Agent Mode)	CFG_RESET_ SOURCE[0:2]	Reset Sequence Duration in CLKIN/PCI_CLK Cycles	Duration
Yes	66 MHz	0	66 MHz	010 (I^2C EEPROM, high PCI_SYNC_IN/ PCI_CLK clock frequency)	37908	568 µs
Yes	66 MHz	1	33 MHz	001 (I^2C EEPROM, low PCI_SYNC_IN/ PCI_CLK clock frequency)	49096/24548	736 µs

4.3.2 Reset Configuration Words

The reset configuration words control the clock ratios and other basic device functions such as PCI host or agent mode, boot location, TSEC modes, and endian mode. The reset configuration words are loaded from the local bus or from the I^2C interface 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 value 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. Modifying values in these other units' memory mapped registers do 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)

The register map for these registers is described in Section 4.5, "Memory Map/Register Definition."

4.3.2.1 Reset Configuration Word Low Register (RCWLR)

The reset configuration word low register is shown in Figure 4-3. This is a read-only register that gets its values according to the reset configuration word low loaded during the reset flow.

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

Table 4-8 defines the reset configuration word low bit fields.

Table 4-8. Reset Configuration Word Low Bit Settings

Bits	Name	Description
0	LBIUCM	Local bus memory controller clock mode. Selects the local bus controller clock ratio. If this bit is set, the local bus controller operates at twice the frequency of the *csb_clk*. If this bit is cleared, the local bus controller operate at the *csb_clk* frequency. The 2:1 mode is useful when the csb_clk operates at low frequency. 0 *csb_clk* ratio is 1:1. 1 *csb_clk* ratio is 2:1.
1	DDRCM	DDR SDRAM memory controller clock mode.Selects the DDR SDRAM memory controller clock ratio. If this bit is set, the DDR SDRAM memory controller operates at twice the frequency of the *csb_clk*. If this bit is cleared, the DDR SDRAM memory controller operates at the *csb_clk* frequency. The 2:1 mode is useful mostly when for a 32-bit data bus memory device. 0 *csb_clk* ratio is 1:1. 1 *csb_clk* ratio is 2:1.
2–3	—	Reserved, should be cleared.
4–7	SPMF	System PLL multiplication factor. See Section 4.3.2.1.1, "System PLL Configuration," 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 *csb_clk* of the device. The encodings for COREPLL are given in the hardware specifications for this device.
16–31	—	Reserved, should be cleared.

4.3.2.1.1 System PLL Configuration

The system PLL ratio reset, shown in Table 4-9, establishes the clock ratio between the CLKIN (PCI host mode) or PCI_CLK (PCI agent mode) input 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)	*csb_clk* : CLKIN (PCI host mode) *csb_clk* : (PCI_CLK x (1+~sampled_cfg_clkin_div))(PCI agent mode)
4–7	SPMF	0000	16 : 1
		0001	Reserved
		0010	2 : 1
		0011	3 : 1
		0100	4 : 1
		0101	5 : 1
		0110	6 : 1
		0111	7 : 1
		1000	8 : 1
		1001	9 : 1
		1010	10 : 1
		1011	11 : 1
		1100	12 : 1
		1101	13 : 1
		1110	14 : 1
		1111	15: 1

NOTE

In PCI host mode, the SPMF field described in Table 4-9 always selects the *csb_clk*:CLKIN ratio regardless of the CFG_CLKIN_DIV reset configuration input value during reset flow.

The SPMF field maximum allowed value is dependent on the value sampled on CFG_CLKIN_DIV during power-on reset, and the LBIUCM and DDRCM reset configuration word fields values. Table 4-10 defines the upper limit of SPMF with respect to these values. Values for SPMF are as follows:

Table 4-10. SPMF Maximum Values

CFG_CLKIN_DIV	LBIUCM	DDRCM	Maximum SPMF Value (decimal)
0	0	0	16
0	0	1	8
0	1	0	8
0	1	1	8
1	0	0	8
1	0	1	4
1	1	0	4
1	1	1	4

4.3.2.2 Reset Configuration Word High Register (RCWHR)

The reset configuration word high is shown in Figure 4-4. This is a read-only register that gets its values according to the reset configuration word high loaded during the reset flow.

	0	1	2	3	4	5	6	7	8	9	11	12	15
Field	PCIHOST	PCI64	PCI1ARB	PCI2ARB	COREDIS	BMS	BOOTSEQ		SWEN	ROMLOC		—	

	16	17	18	19	20	27	28	29	30	31
Field	TSEC1M		TSEC2M		—		TLE	LALE	LDP	—

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

Table 4-11 defines the reset configuration word high bit fields.

Table 4-11. Reset Configuration Word High Bit Settings

Bits	Name	Description
0	PCIHOST	PCI host mode. See Section 4.3.2.2.1, "PCI Host/Agent Configuration," for more information.
1	PCI64	PCI 64-bit bus mode. Configures the device to have a 64-bit PCI interface. In this mode, only the PCI1 interface is enabled. For details, see the PCI programming models described in Section 13.3.3.6, "Standard Programming Interface Configuration Register." 0 The device uses a 32-bit PCI interface. In host mode, both PCI1 and PCI2 are enabled. 1 The device uses a 64-bit PCI interface. Only PCI1 is enabled. The value of PCI64 also defines the function of the PCI2 interface pins, as follows:

Pin Function When PCI64 = 0	Pin Function When PCI64 = 1	Pin Function When PCI64 = 0	Pin Function When PCI64 = 1
$\overline{\text{PCI2_RESET_OUT}}$	GPIO20	$\overline{\text{PCI2_STOP}}$	GPIO2[4]
PCI2_AD[31:0]	PCI1[63:32]	$\overline{\text{PCI2_DEVSEL}}$	GPIO2[5]
PCI2_C/$\overline{\text{BE}}$[3:0]	PCI1_C/$\overline{\text{BE}}$[7:4]	$\overline{\text{PCI2_SERR}}$	$\overline{\text{PCI1_ACK64}}$
PCI2_PAR	PCI1_PAR64	$\overline{\text{PCI2_PERR}}$	$\overline{\text{PCI1_REQ64}}$
$\overline{\text{PCI2_FRAME}}$	GPIO2[1]	$\overline{\text{PCI2_REQ}}$[0:2]	GPIO2[6:8]
$\overline{\text{PCI2_TRDY}}$	GPIO2[2]	$\overline{\text{PCI2_GNT}}$[0:2]	GPIO2[9:11]
$\overline{\text{PCI2_IRDY}}$	GPIO2[3]		

Table 4-11. Reset Configuration Word High Bit Settings (continued)

Bits	Name	Description
2	PCI1ARB	PCI1 internal arbiter mode. Enables the on-chip PCI1 arbiter. 0 On-chip PCI1 arbiter is disabled. External arbitration is required. 1 On-chip PCI1 arbiter is enabled. The value of PCI1ARB also defines the function of the PCI1 arbitration signals that are multiplexed with CompactPCI signals, as follows: **Pin Function When PCI1ARB = 0** / **Pin Function When PCI1ARB = 1** CPCI_HS_ES / $\overline{\text{PCI1_REQ}}$[1] CPCI_HS_LED / $\overline{\text{PCI_GNT}}$[1] CPCI_HS_ENUM / $\overline{\text{PCI_GNT}}$[2]
3	PCI2ARB	PCI2 internal arbiter mode. Enables the internal PCI2 arbiter. 0 On-chip PCI2 arbiter is disabled. External arbitration is required. 1 On-chip PCI2 arbiter is enabled.
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 Section 6.2.1, “Arbiter Configuration Register (ACR).” 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 holdoff mode. The core is prevented from booting until it is configured by an external master.
5	BMS	Boot memory space. See Section 4.3.2.2.2, “Boot Memory Space (BMS),” for more information.
6–7	BOOTSEQ	Boot sequencer configuration. See Section 4.3.2.2.3, “Boot Sequencer Configuration,” 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. See Section 4.3.2.2.4, “Boot ROM Location for more information.
12–15	—	Reserved, should be cleared.
16–17	TSEC1M	TSEC1 mode. See Section 4.3.2.2.5, “TSEC1 Mode for more information.
18–19	TSEC2M	TSEC2 mode. See Section 4.3.2.2.6, “TSEC2 Mode for more information.
20–27	—	Reserved, should be cleared.
28	TLE	True little-endian. See Section 4.3.2.2.7, “e300 Core True Little-Endian for more information.
29	LALE	Local bus LALE signal timing. See Section 4.3.2.2.8, “LALE Configuration for more information.
30	LDP	LDP pin mux state after reset. See Section 4.3.2.2.9, “LDP Configuration for more information.
31	—	Reserved, should be cleared.

4.3.2.2.1 PCI Host/Agent Configuration

The PCIHOST configuration parameter, shown in Table 4-12, configures the device to act as a PCI host or as a PCI agent device. In host mode, both PCI1 and PCI2 interfaces are enabled, and , the device can immediately master transactions to the PCI interfaces. If the device is a PCI agent device,only PCI1 interface is enabled (PCI2 can not be used), and the device is disabled from mastering PCI transactions until the external host enables it to do so. The external host does this by setting the control registers of the device's interfaces appropriately. See details in the PCI programming model described in Section 13.3, "Memory Map/Register Definitions."

Table 4-12. PCI Host/Agent Configuration

RCWHR Bit	Field Name	Value (Binary)	Meaning
0	PCIHOST	0	The device acts as a PCI agent device. Only PCI1 is enabled.
		1	The device acts as the host processor (default). Both PCI1 and PCI2 are enabled.

NOTE

If the device is a PCI agent, and the e300 core is not in holdoff mode (as described in Section 4.3.2.2, "Reset Configuration Word High Register (RCWHR)"), the boot ROM should not be located on the PCI1 interface because the device is not enabled to master reads onto the PCI bus.

4.3.2.2.2 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 begins fetching boot code from one of two addresses: 0x0000_0100 or 0xFFF0_0100, and exceptions are vectored to physical addresses 0x000*n_nnnn* or 0xFFF*n_nnnn* appropriately. This bit specifies whether an interrupt vector offset is prepended with 0xFFF 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 4-13, specifies both the device boot ROM address window and the initial e300 core boot address.

Table 4-13. 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 0x000*n_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 0xFFF0_0100 and exceptions are vectored to the physical address of 0xFFF*n_nnnn*.

4.3.2.2.3 Boot Sequencer Configuration

The boot sequencer configuration options, shown in Table 4-14, allow the boot sequencer to load configuration data from the serial ROM located on the I^2C port before the host tries to configure the device. These options also specify normal or extended I^2C addressing modes. See Section 17.4.5, "Boot Sequencer Mode."

Table 4-14. Boot Sequencer Configuration

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

4.3.2.2.4 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 4-15, establishes the location of boot ROM. Accesses to the boot vector and the default boot ROM region of the local address map are directed to the interface specified by this field.

Table 4-15. Boot ROM Location

RCWHR Bits	Field Name	Value (Binary)	Meaning
9–11	ROMLOC	000	DDR SDRAM
		001	PCI1
		010	PCI2
		011	Reserved, should be cleared.
		100	Reserved
		101	Local bus GPCM—8-bit ROM
		110	Local bus GPCM—16-bit ROM
		111	Local Bus GPCM—32-bit ROM

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 5.2, “Local Memory Map Overview and Example.”

NOTE

In PCI host mode, although selecting the PCI1 or PCI2 option with ROMLOC set the appropriate local access window, $\overline{\text{PCI1_RESET_OUT}}$ and $\overline{\text{PCI2_RESET_OUT}}$ remain asserted and PCI_CLK_OUT[x] are disabled after reset.

In this case 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).” The boot sequencer should write to the appropriate registers to negate $\overline{\text{PCI1_RESET_OUT}}$ or $\overline{\text{PCI2_RESET_OUT}}$ and enable the appropriate clocks to the PCI ROM device. Only then should it enable the boot vector fetch by clearing ACR[COREDIS] as described in Section 6.2.1, “Arbiter Configuration Register (ACR).”

4.3.2.2.5 TSEC1 Mode

The TSEC1 mode reset configuration word field, shown in Table 4-16, selects standard versus reduced width and the protocol used by the TSEC1 controller (three-speed Ethernet controller interface).

Table 4-16. TSEC1 Mode Configuration

Reset Configuration Word High Register (RCWHR) Bits	Field Name	Value (Binary)	Meaning
16–17	TSEC1M	00	The TSEC1 controller operates in the RGMII protocol, using only four transmit data signals and four receive data signals.
		01	The TSEC1 controller operates in the RTBI protocol, using only four transmit data signals and four receive data signals.
		10	The TSEC1 controller operates in the GMII protocol, using eight transmit data signals and eight receive data signals.
		11	The TSEC1 controller operates in the TBI protocol, using eight transmit data signals and eight receive data signals.

NOTE

The reset value of the system I/O configuration register high (SICRH) depends on the reset configuration word high TSEC1M field setting. This is used to avoid contention in systems not using the TBI or RTBI modes. In non-TBI modes, device signals which have additional functions are set to be in a non-TSEC function, thus not driven during and after reset. The function of these signals can be changed by writing to this register during system initialization. See Section 5.3.2.6, “System I/O Configuration Register High (SICRH).”

4.3.2.2.6 TSEC2 Mode

The TSEC2 mode reset configuration word field, shown in Table 4-17, selects standard versus reduced width and the protocol used by the TSEC2 controller (three-speed Ethernet controller interface).

Table 4-17. TSEC2 Mode Configuration

Reset Configuration Word High Register (RCWHR) Bits	Field Name	Value (Binary)	Meaning
18–19	TSEC2M	00	The TSEC2 controller operates in the RGMII protocol, using only four transmit data signals and four receive data signals.
		01	The TSEC2 controller operates in the RTBI protocol, using only four transmit data signals and four receive data signals.
		10	The TSEC2 controller operates in the GMII protocol, using eight transmit data signals and eight receive data signals.
		11	The TSEC2 controller operates in the TBI protocol, using eight transmit data signals and eight receive data signals.

NOTE

The reset value of the system I/O configuration register high (SICRH) depends on the setting of TSEC2M. This is used to avoid contention in systems not using TBI or RTBI modes. In non-TBI modes, device signals which have additional functions are set to be in a non-TSEC function, thus not driven during and after reset. The function of these signals can be changed by writing to this register during system initialization. See Section 5.3.2.6, “System I/O Configuration Register High (SICRH).”

4.3.2.2.7 e300 Core True Little-Endian

The true little endian reset configuration word field, shown in Table 4-18, selects whether the e300 core operates in big-endian mode or true little-endian mode at reset.

Table 4-18. 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

4.3.2.2.8 LALE Configuration

The LALE reset configuration word field, shown in Table 4-19, configures the timing of the local bus LALE signal. Refer to the hardware specifications for specific timing information.

Table 4-19. LALE Configuration

Reset Configuration Word High Register (RCWHR) Bit	Field Name	Value (Binary)	Meaning
29	LALE	0	Normal LALE timing.
		1	LALE is negated 1/2 an LBC_controller_clk earlier.

4.3.2.2.9 LDP Configuration

The LDP reset configuration word field configures the initial state of SICRL[LDP_A], which controls the functionality of the LPD[0:3] pins. Table 4-20 shows the LDP configuration.

Table 4-20. LDP Configuration

Reset Configuration Word High Register (RCWHR) Bit	Field Name	Meaning
30	LDP	0 Initial value of SICRL[LDP_A] is 1, meaning that LDP0–LDP3 are used for local data parity. 1 Initial value of SICRL[LDP_A] is 0; the meaning is as follows: • LDP0 is used as $\overline{\text{CKSTOP_OUT}}$ • LDP1 is used as $\overline{\text{CKSTOP_IN}}$. • LDP2 is used as LCS4 • LDP3 is used as LCS5

4.3.3 Loading the Reset Configuration Words

The device loads the reset configuration words from a local bus EEPROM, or an I^2C 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 in detail.

4.3.3.1 Loading from Local Bus EEPROM

The reset configuration words are assumed to reside in an EEPROM 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 4-21 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 4-21 are always read on byte lane LAD[0:7] regardless of the port size.

Table 4-21. 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

The device first reads a value from address 0x00 then reads a value from addresses 0x08, 0x10, and 0x18. These four bytes are used to form the reset configuration word low. It then proceeds reading the bytes from addresses 0x20, 0x28, 0x30, and 0x38, which form the reset configuration word high.

Table 4-22 shows the data structure of the local bus device containing the reset configuration words (RCWL and RCWH).

Table 4-22. 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 EEPROM Timing

For loading the reset configuration word from a local bus EEPROM, the PCI_SYNC_IN/PCI_CLK input clock is divided by 32 to enable operation of slow frequency memories. Figure 4-5 and Figure 4-6 show the timing of EEPROM operation.

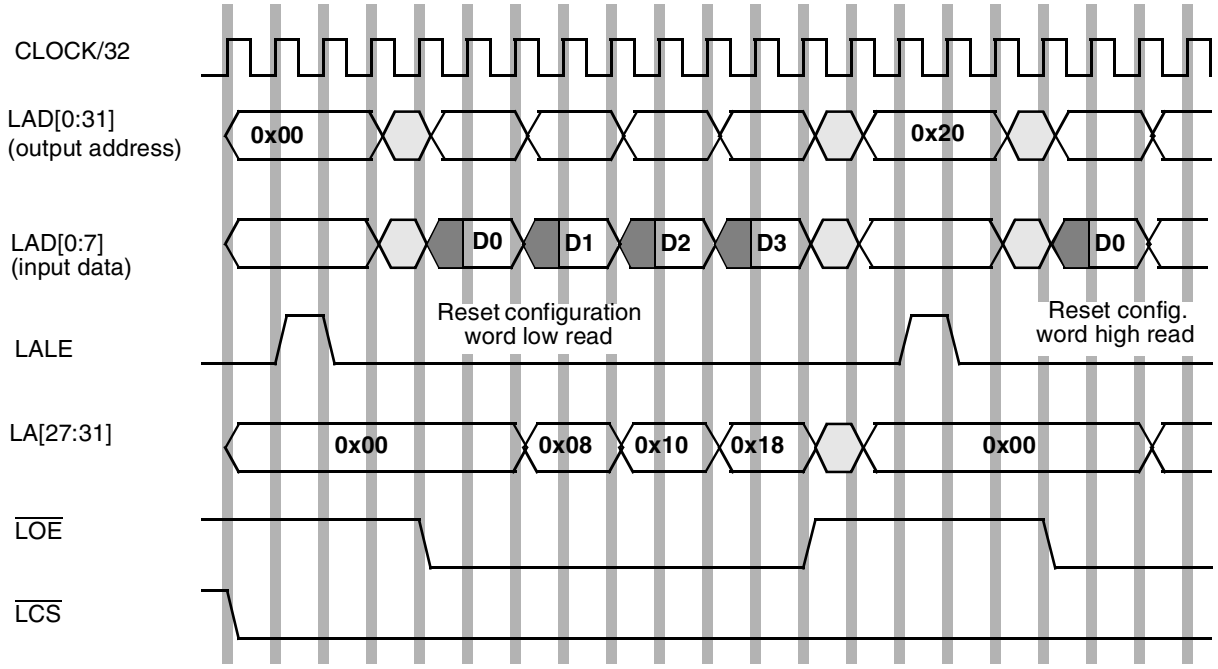


Figure 4-5. Loading Reset Configuration Words from Local Bus

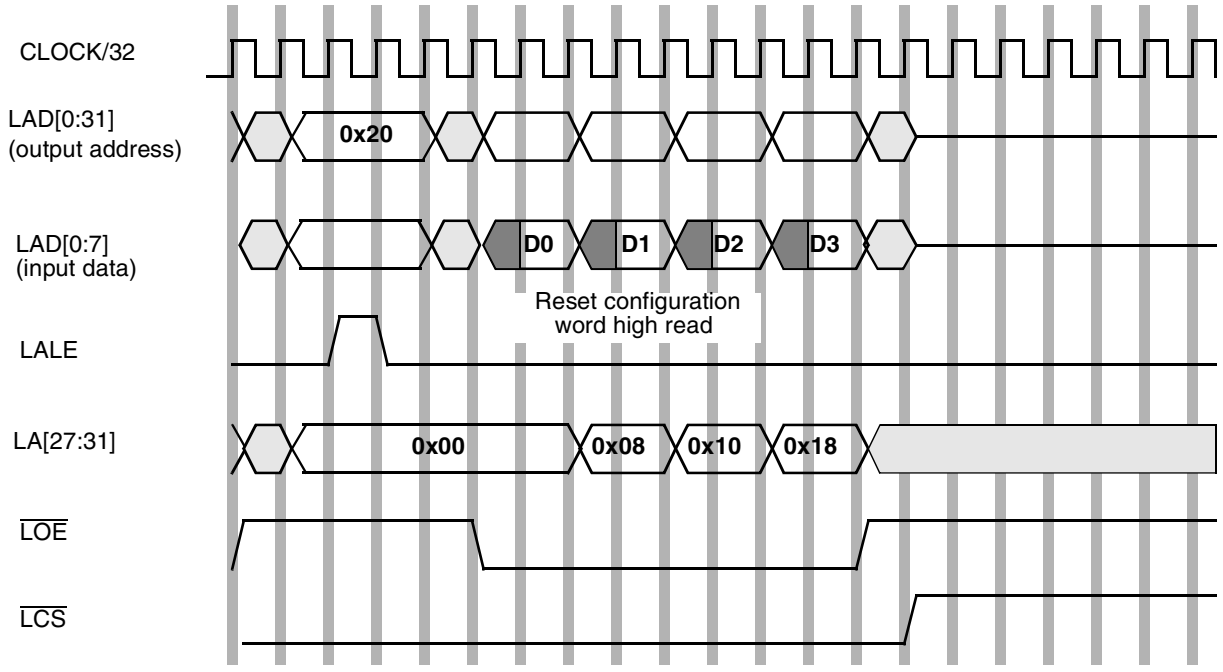


Figure 4-6. Loading Reset Configuration Words from Local Bus (continued)

4.3.3.2 Loading from I²C EEPROM

The device is capable of loading the reset configuration word from the I²C interfaces (the device has two I²C interfaces, but only I²C #1 can be used for this purpose). If the device is configured to load the reset configuration word from the I²C interface, according to the reset configuration input signals, it uses the I²C unit boot sequencer in a special mode. In this mode, the I²C 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 I²C serial EEPROM.

Note that this does not prevent using the I²C 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 a detailed description about the I²C interface and the boot sequencer refer to Section 17.4.5, “Boot Sequencer Mode.”

NOTE

When boot sequencer mode is used an I²C serial EEPROM of extended addressing type must be used.

If the I²C interface is used for loading the reset configuration words, the I²C 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 I²C module enters its reset state until $\overline{\text{HRESET}}$ is negated. There should be no other I²C traffic when the boot sequencer is active.

After $\overline{\text{HRESET}}$ is negated, the functional boot sequencer, in extended I²C 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 I²C module expects that a particular data format be used for data in the EEPROM. A preamble should be the first 3 bytes programmed into the EEPROM. It should have a value of 0xAA55AA. The I²C 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 4-7.

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^2C 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 4-7. EEPROM Data Format for Reset Configuration Words Preload Command

Figure 4-8 shows an example of the EEPROM contents, including the preamble, reset configuration words and additional initialization data, and CRC. 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	
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	
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 4-8. EEPROM Contents

4.3.3.2.4 Reset Configuration Load Fail

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

NOTE

The MPC8349/47/43 silicon rev 1. does not attempt to reload the HRCW as described above, instead it will hang pulling the I2C signals low until a power-on reset occurs (HRESET asserted, and most of the I/O drivers are disabled).

4.3.3.3 Default Reset Configuration Words

If the device is configured not to load the reset configuration words from a local bus EEPROM or I^2C EEPROM, it can also be initialized with one of five hard-coded default options, selected by the reset configuration input signals, CFG_RESET_SOURCE[0:2]. In this mode, the device is assumed to be a PCI agent, and therefore only clock modes differ among the four options.

The reset configuration words are driven internally with the values shown in Table 4-23 and Table 4-24.

NOTE

In this mode the device is also configured to accept PCI configuration cycles when completing its reset sequence (In PCI function configuration register, the CFG_LOCK bit is cleared). In addition, the inbound window size of the PCI inbound window attribute registers (PIWAR*n*[IWS]) is set to 0b010011, defining 1-Mbyte ($2^{(19+1)}$) memory windows. See Section 13.3.3.24, "PCI Function Configuration Register."

4.3.3.3.1 Hard-Coded Reset Configuration Word Low

Table 4-23 defines the hard-coded reset configuration word low fields values, according to the CFG_RESET_SOURCE[0:2].

Table 4-23. Hard-Coded Reset Configuration Word Low Fields Values

Bits	Name	CFG_RESET_SOURCE Value					Meaning
		011	100	101	110	111	
0	LBIUCM	0	0	1	1	0	LBC controller clock: *csb_clk* 0 1:1 1 2:1
1	DDRCM	0	0	1	1	0	DDR controller clock: *csb_clk* 0 1:1 1 2:1
2–3	Res	10	10	10	10	10	—

Table 4-23. Hard-Coded Reset Configuration Word Low Fields Values (continued)

Bits	Name	CFG_RESET_SOURCE Value					Meaning
		011	100	101	110	111	
4–7	SPMF	0100	1000	0100	0101	0100	*csb_clk*: PCI_CLK ratio SPMF:1
8	Res	0	0	0	0	0	—
9–15	COREPLL	0100011	0100011	0100100	0100100	0000100	Core clock: *csb_clk* ratio
16–31	Res	0x0006	0x000C	0x0008	0x000A	0x0006	—

4.3.3.3.2 Hard-Coded Reset Configuration Word High Fields Values

Table 4-24 defines the hard-coded reset configuration word high fields values. Note that the values put into the fields RCWHR are the same for all values of CFG_RESET_SOURCE[0:2] from 011 to 111; these values select hard-coded reset configuration words options, as described in Section 4.3.1.1, "Reset Configuration Word Source."

Table 4-24. Hard-Coded Reset Configuration Word High Field Values

Bits	Name	Field Value	Meaning
CFG_RESET_SOURCE[0:2] = 011–111			
0	PCIHOST	0	PCI agent mode
1	PCI64	MPC8349: 1; MPC8347/MPC8343: 0	PCI 64-bit bus mode
2	PCI1ARB	0	External arbiter is used.
3	PCI2ARB	0	—
4	COREDIS	1	e300 core is disabled (boot holdoff)
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	000	Boot ROM interface location.
12–15	Reserved, should be cleared.	0000	—
16–17	TSEC1M	10	GMII mode
18–19	TSEC2M	10	GMII mode
20–27	Reserved, should be cleared.	0000_0000	—
28	TLE	0	Big endian mode
29	LALE	0	Normal timing
30	LDP	0	LPD pins used for local data parity
31	Reserved	0	—

4.3.3.3.3 Examples for Hard-Coded Reset Configuration Words Usage

Examples for various clock modes are listed in Table 4-25.

Table 4-25. Examples For Hard-Coded Reset Configuration Words Usage

CFG_RESET_SOURCE[0:2]	PCI_CLK (MHz)	*csb_clk* (MHz)	Core Clock (MHz)
011	66	266	400
100	33	266	400
101	33	133	266
110	33	166	333
111	66	266	533

4.4 Clocking

Figure 4-9 shows the internal distribution of clocks within the device.

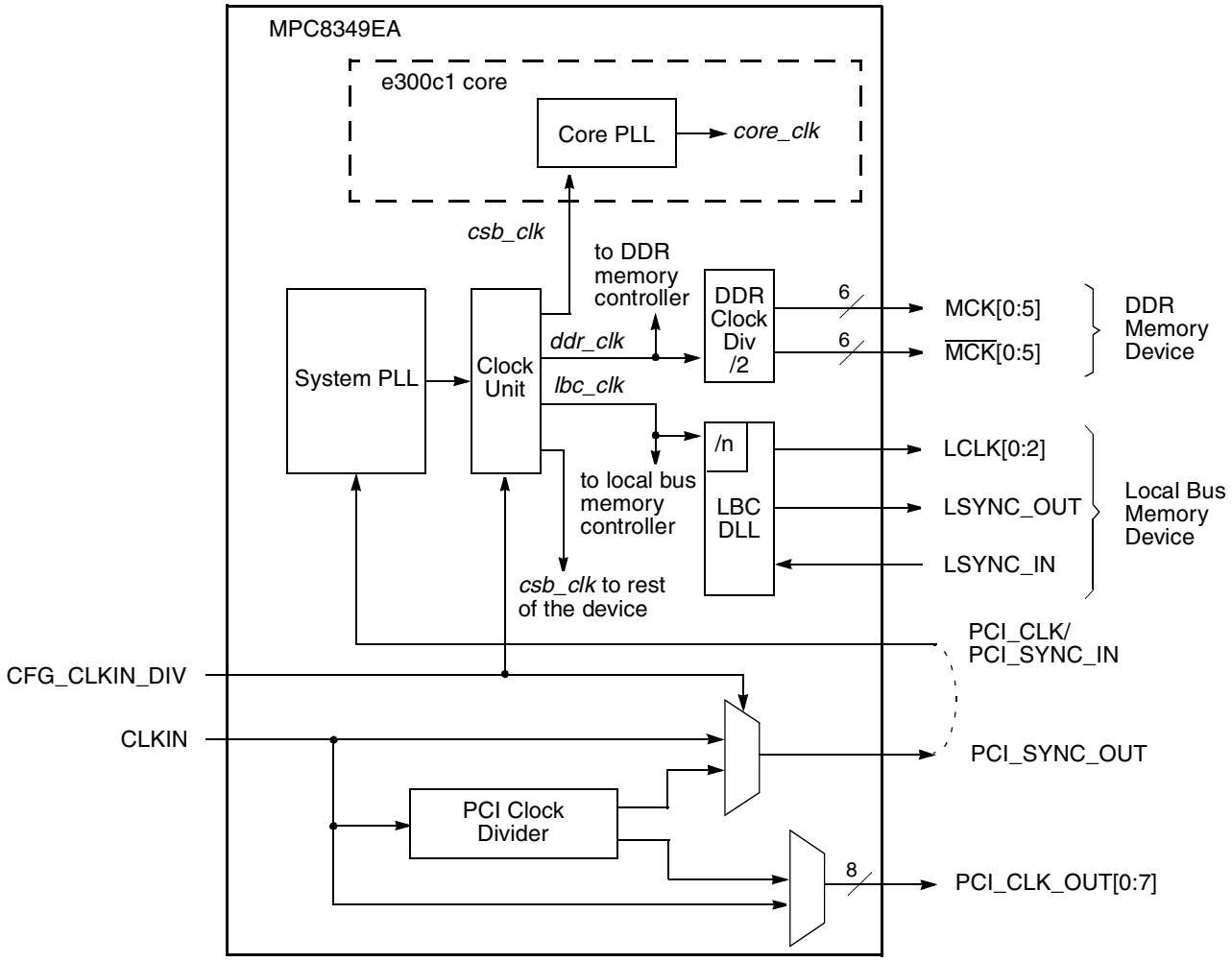


Figure 4-9. Clock Subsystem Block Diagram

The primary clock source for the device can be one of two inputs, CLKIN or PCI_CLK, depending on whether the device is configured in PCI host or PCI agent mode, respectively.

NOTE

The MPC8347E and MPC8343E only support five PCI clock output signals (PCI_CLK_OUT[0:4]).

4.4.1 Clocking in PCI Host Mode

When the device is configured as a PCI host device (RCWH[PCIHOST] = 1), CLKIN is the primary input clock. CLKIN feeds the PCI clock divider (÷2) and the PCI_SYNC_OUT and PCI_CLK_OUT multiplexors. The CFG_CLKIN_DIV configuration input selects whether CLKIN or CLKIN/2 is driven out on the PCI_SYNC_OUT signal.

PCI_SYNC_OUT is connected externally to PCI_SYNC_IN to allow the internal clock subsystem to synchronize to the system PCI clocks. PCI_SYNC_OUT must be connected properly to PCI_SYNC_IN, with equal delay to all PCI agent devices in the system, to allow the MPC8349EA to function.

4.4.1.1 PCI Clock Outputs (PCI_CLK_OUT[0:7])

When the device is configured as a PCI host , it provides eight clock output signals, PCI_CLK_OUT[0:7], for external PCI agents. Each clock output can be configured independently by a memory-mapped register. The OCCR[PCICD*n*] parameters select whether CLKIN or CLKIN/2 is driven out on the PCI_CLK_OUT*n* signals. See Section 4.5.2.2, “Output Clock Control Register (OCCR).”

When the device comes out of reset, the PCI clock outputs are disabled and are actively driven to a steady low state. Each of the individual clock outputs can be enabled (enable toggling of the clock) by setting its corresponding OCCR[PCICOE*n*] bit. All output clocks are phase aligned to each other and to PCI_SYNC_OUT.

4.4.2 Clocking In PCI Agent Mode

When the device is configured as a PCI agent, PCI_CLK is the primary input clock. In agent mode, the CLKIN signal should be tied to GND, and the clock output signals, PCI_CLK_OUT*n* and PCI_SYNC_OUT, are not used.

In agent mode, the CFG_CLKIN_DIV configuration input can be used to double the internal clock frequencies, if sampled as 1 during PORESET assertion. This feature is useful if a fixed internal frequency is desired regardless of whether the PCI clock is running at 33 or 66 MHz. PCI specifications require that the signal M66EN provides the PCI clock frequency information.

4.4.3 System Clock Domains

As shown in Figure 4-9, the primary clock input (PCI_CLK/PCI_SYNC_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 from a complex set of factors that can be simplified into the following equation:

$$csb_clk = [\text{PCI_SYNC_IN} \times (1 + \overline{\text{CFG_CLKIN_DIV}})] \times \text{SPMF}$$

In PCI host mode, PCI_SYNC_IN × (1 + $\overline{\text{CFG_CLKIN_DIV}}$) is the CLKIN frequency.

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."

Section 4.3.2.1, "Reset Configuration Word Low Register (RCWLR)The internal *ddr_clk* frequency is determined by RCWL[DDRCM]. See Section 4.3.2.1, "Reset Configuration Word Low Register (RCWLR)." Note that *ddr_clk* is not the external memory bus frequency; *ddr_clk* passes through the DDR clock divider (÷2) to create the differential DDR memory bus clock outputs (MCK and $\overline{\text{MCK}}$). However, the data rate is the same frequency as *ddr_clk*.

The internal *lbc_clk* frequency is determined by RCWL[LBIUCM]. See Section 4.3.2.1, "Reset Configuration Word Low Register (RCWLR)." 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 outputs (LSYNC_OUT and LCLK[0:2]). The LBC clock divider ratio is controlled by LCCR[CLKDIV]. See Section 10.1.2.1, "LBC 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 4-26 specifies which units have a configurable clock frequency. Refer to Section 4.5.2.3, "System Clock Control Register (SCCR)."

Table 4-26. Configurable Clock Units

Unit	Default Frequency	Options
TSEC1	*csb_clk*/3	Off, *csb_clk*, *csb_clk*/2, *csb_clk*/3
TSEC2, I^2C1	*csb_clk*/3	Off, *csb_clk*, *csb_clk*/2, *csb_clk*/3
Security core	*csb_clk*/3	Off, *csb_clk*, *csb_clk*/2, *csb_clk*/3
USB MPH	*csb_clk*/3	Off, *csb_clk*/2, *csb_clk*/3
USB DR	*csb_clk*/3	Off, *csb_clk*,*csb_clk*/2, *csb_clk*/3
PCI1, PCI2 and DMA complex	*csb_clk*	Off, *csb_clk*

NOTE

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

TSEC2 and I^2C1share the same clock control, and therefore the same clock ratio. It is not possible to operate any of these units when the clock is disabled.

USB MPH and USB DR units must have the same clock ratio, with the exception when one of the unit's clock is off. If one unit is disabled, the other unit can operate in any of the possible ratios.

4.5 Memory Map/Register Definition

4.5.1 Reset Configuration Registers Descriptions

The reset configuration and status registers are shown in Table 4-27.

Table 4-27. Reset Configuration and Status Registers Memory Map

Address	Use	Access	Section/Page
0x0_0900	Reset configuration word low register (RCWLR)	R	4.5.1.1/4-32
0x0_0904	Reset configuration word high register (RCWHR)	R	4.5.1.2/4-32
0x0_0908	Reserved, should be cleared.	—	—
0x0_090C	Reserved, should be cleared.	—	—
0x0_0910	Reset status register (RSR)	R/W	4.5.1.3/4-33
0x0_0914	Reset mode register (RMR)	R/W	4.5.1.4/4-34
0x0_0918	Reset protection register (RPR)	R/W	4.5.1.5/4-35
0x0_091C	Reset control register (RCR)	R/W	4.5.1.6/4-36
0x0_0920	Reset control enable register (RCER)	R/W	4.5.1.7/4-36
0x0_0924–0x0_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)

The RSR shown in Figure 4-10 captures various reset events in the device.

Address 0x0_0910 Access: User read/write

	0	2	3									14	15
R W	RSTSRC		—										BSF
Reset	n1		0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	22	23	24	26	27	28	29	30	31
R W	—		SWSR	SWHR	—		JSRS	—		CSHR	SWRS	BMRS	SRS	HRS
Reset	All zeros													

1 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 4-10. Reset Status Register (RSR)

Table 4-28 defines the reset status register bit fields.

Table 4-28. Reset Status Register Field Descriptions

Bits	Name	Description
0–2	RSTSRC	Reset configuration word source. Reflects the value of CFG_RESET_SOURCE input signal during the reset flow. See Section 4.3.1.1, “Reset Configuration Word Source,” on page 4-9 for detailed description. Changing this field has no effect.
3–14	—	Reserved, should be cleared.
15	BSF	Boot sequencer fail. If set, indicates that the I^2C boot sequencer has failed while loading the reset configuration words. Cleared by writing a 1 to it (writing zero has no effect).
16–17	—	Reserved, should be cleared.
18	SWSR	Software soft reset. If set, indicates that a software soft reset has occurred. Cleared by writing a 1 to it (writing zero has no effect).
19	SWHR	Software hard reset. If set, indicates that a software hard reset has occurred. Cleared by writing a 1 to it (writing zero has no effect).
20–22	—	Reserved, should be cleared.
23	JSRS	JTAG soft reset status. When the JTAG reset request is set, JTRS is set and remains set until software clears it. JSRS is cleared by writing a 1 to it (writing zero has no effect). 0 No JTAG reset event occurred 1 A JTAG reset event occurred
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 occurred 1 An enabled check stop reset event occurred

Table 4-28. 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 one to it (writing zero has no effect). 0 No software watchdog reset event occurred 1 A software watchdog reset event has occurred
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 has occurred. 1 A bus monitor reset event has occurred.
30	SRS	Soft reset status. When an external or internal soft reset event is detected, SRS is set and remains that way until software clears it. SRS is cleared by writing a 1 to it (writing zero has no effect). 0 No soft reset event has occurred. 1 A soft reset event has occurred.
31	HRS	Hard reset status. When an external or internal hard reset event is detected, HRS is set and remains that way until software clears it. HRS is cleared by writing a 1 (writing zero has no effect). 0 No hard reset event has occurred. 1 A hard reset event has occurred.

NOTE

The reset status register accumulates reset events. For example, because software watchdog expiration results in a hard reset, which in turn results in a soft reset, RSR[SWRS], RSR[SRS], and RSR[HRS] are all set after a software watchdog reset. This register returns to its reset value only when power-on reset occurs.

4.5.1.4 Reset Mode Register (RMR)

The reset mode register (RMR), shown in Figure 4-11, is used to enable a hard reset sequence on the device whenever the e300 core enters checkstop state.

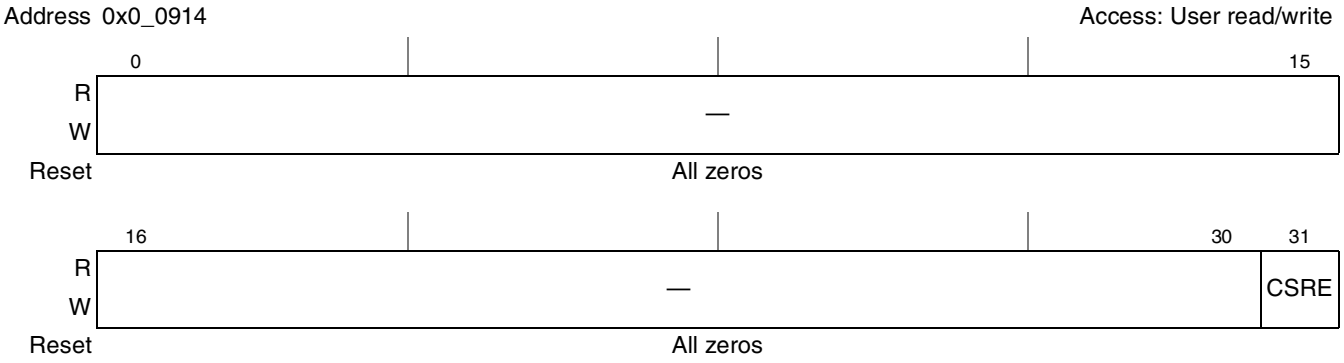


Figure 4-11. Reset Mode Register (RMR)

Table 4-29 describes RMR fields.

Table 4-29. 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 chip to perform a hard reset sequence whenever 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)

The reset protection register, shown in Figure 4-12, is used to prevent unintended software reset requests caused by writes to the reset control register (RCR). The user should write the value 0x52535445 (RSTE in ASCII) to enable. Enable indication appears in the reset control enable register (RCER[CRE]). Reading this register always returns all zeros. To disable write to the reset control register (RCR), the user should write one to RCER[CRE].

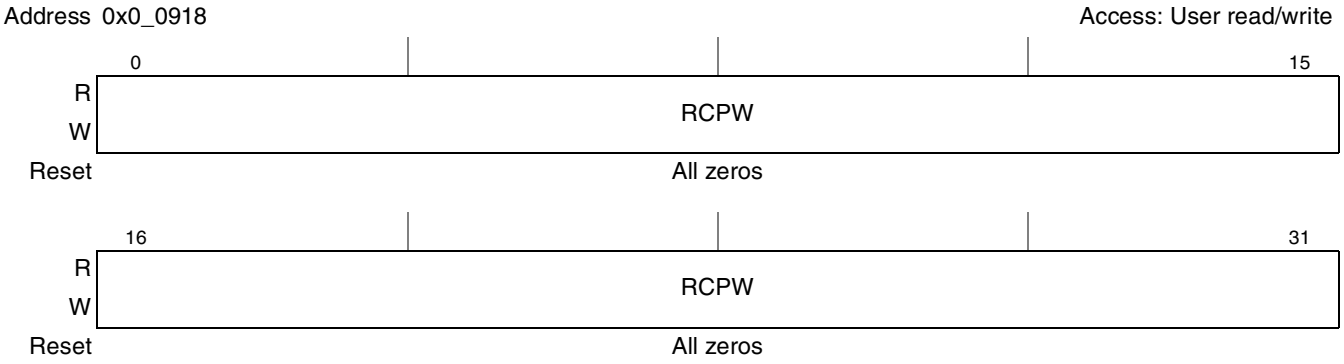


Figure 4-12. Reset Protection Register (RPR)

Table 4-30 defines the bit fields of RPR.

Table 4-30. RPR Bit Settings

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)

The RCR, shown in Figure 4-13, can be used by software to initiate a soft or hard reset sequence. To allow writing to this register, the user must enable it by writing the value 0x5253_5445 to the RPR.

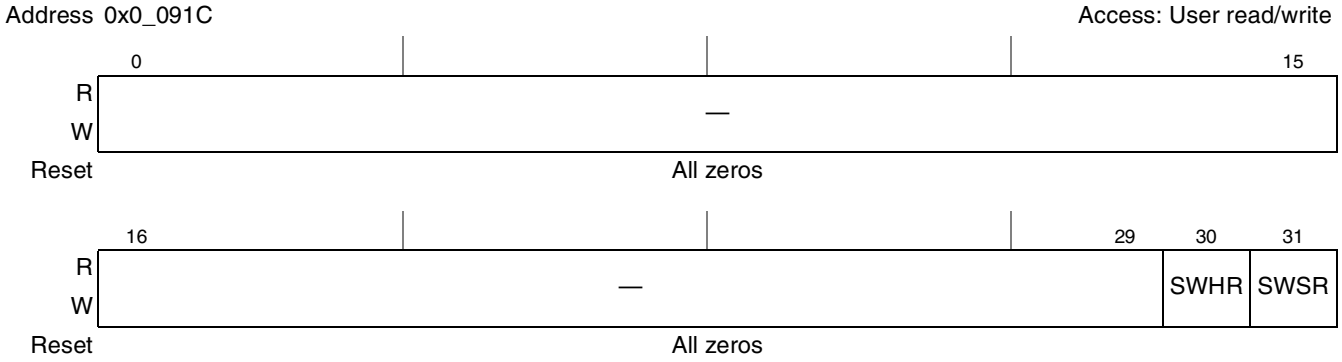


Figure 4-13. Reset Control Register (RCR)

Table 4-31 defines the bit fields of RCR.

Table 4-31. RCR Bit Settings

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

4.5.1.7 Reset Control Enable Register (RCER)

The RCER shown in Figure 4-14, indicates by the CRE field that the RPR is accessed with a value that enables RCR.

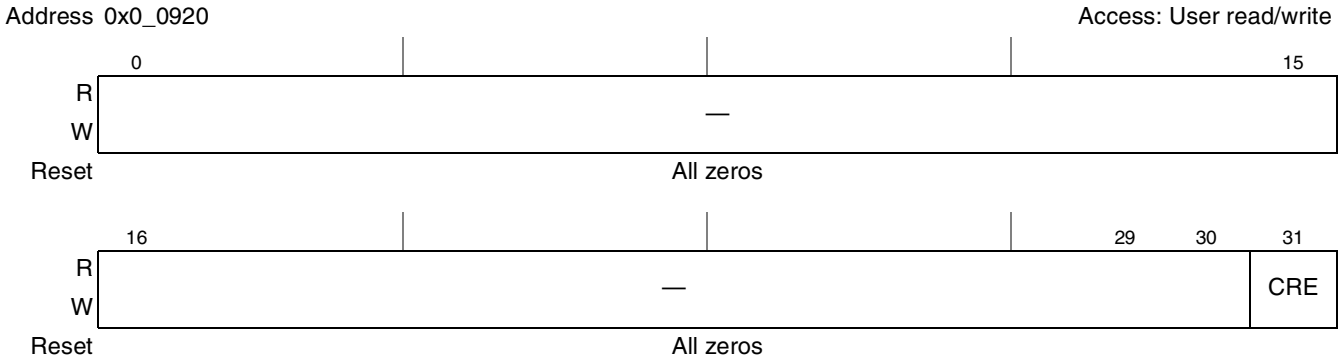


Figure 4-14. Reset Control Enable Register (RCER)

Table 4-32 defines the bit fields of RCER.

Table 4-32. 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 4-33.

Table 4-33. Clock Configuration Registers Memory Map

Address	Use	Access	Section/Page
0x0_0A00	System PLL mode register (SPMR)	R	4.5.2.1/4-37
0x0_0A04	Output clock control register (OCCR)	R/W	4.5.2.2/4-38
0x0_0A08	System clock control register (SCCR)	R/W	4.5.2.3/4-40
0x0_0A0C–0x0_0AFC	Reserved, should be cleared.	—	—

4.5.2.1 System PLL Mode Register (SPMR)

The SPMR is shown in Figure 4-15. This read-only register gets its values according to the CFG_CLKIN_DIV 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 a hard reset sequence. It may hold values different than those in the RCWLR after a a hard reset sequence.

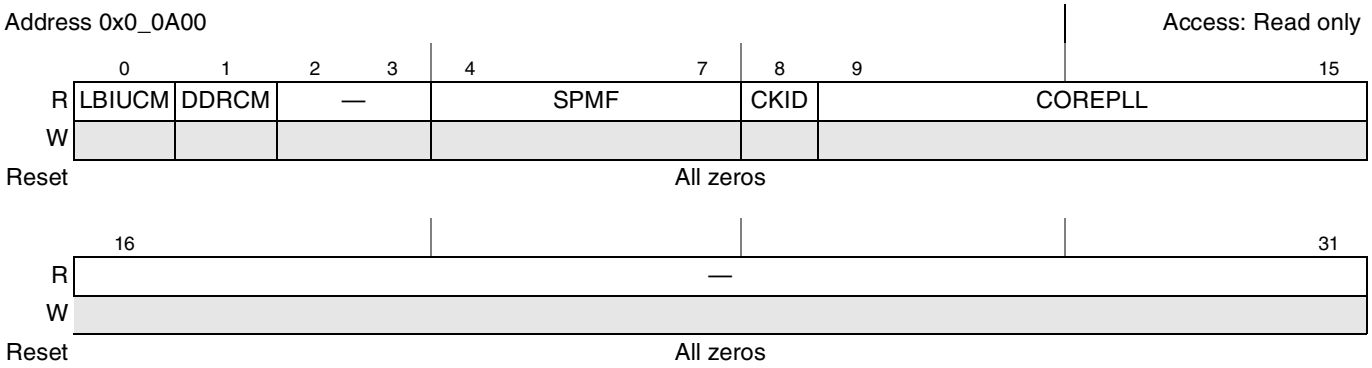


Figure 4-15. System PLL Mode Register

Table 4-34 defines the system PLL mode register bit fields.

Table 4-34. System PLL Mode Register Bit Settings

Bits	Name	Meaning	Description
0	LBIUCM	Local bus memory controller clock mode.	Section 4.3.2.1, “Reset Configuration Word Low Register (RCWLR)”
1	DDRCM	DDR SDRAM memory controller clock mode.	Section 4.3.2.1, “Reset Configuration Word Low Register (RCWLR)”
2–3	—	Reserved, should be cleared.	—
4–7	SPMF	System PLL multiplication factor	Section 4.3.2.1.1, “System PLL Configuration”
8	CKID	CLKIN division factor. Reflects the value of CFG_CLKIN_DIV input signal during the reset flow.	Section 4.3.1.2, “CLKIN Division”
9–15	COREPLL	Core PLL configuration.	See MSC7120 Hardware Specifications.
16–31	—	Reserved, should be cleared.	—

4.5.2.2 Output Clock Control Register (OCCR)

The OCCR shown in Figure 4-16, controls the device output clocks. It is possible to control some output clock modes by writing to this memory mapped register as described below.

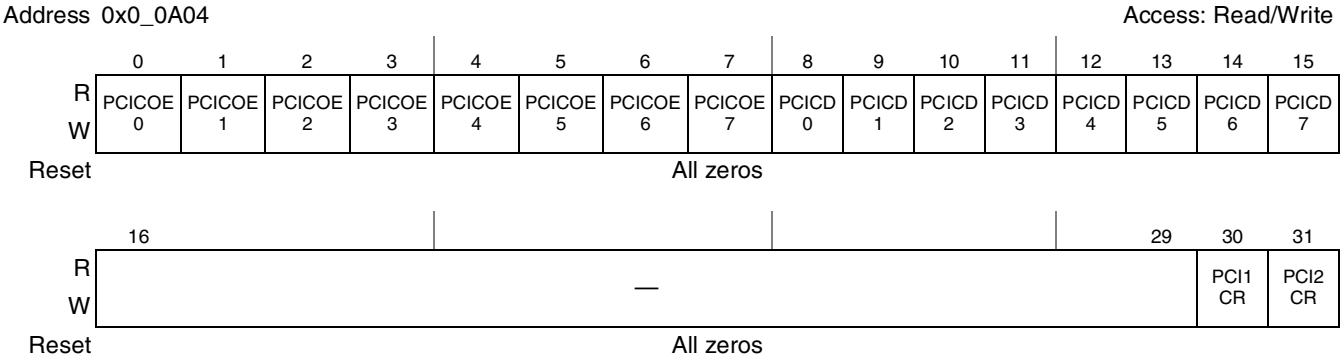


Figure 4-16. Output Clock Control Register (OCCR)

Table 4-35 defines the bit fields of OCCR.

Table 4-35. OCCR Bit Settings

Bits	Name	Description
0	PCICOE0	PCI_CLK_OUT0 enable. 0 PCI_CLK_OUT0 signal is disabled (drive constant ‘0’). 1 PCI_CLK_OUT0 signal is enabled to toggle.
1	PCICOE1	PCI_CLK_OUT1 enable. 0 PCI_CLK_OUT1 signal is disabled (drive constant ‘0’). 1 PCI_CLK_OUT1 signal is enabled to toggle.

Table 4-35. OCCR Bit Settings (continued)

Bits	Name	Description
2	PCICOE2	PCI_CLK_OUT2 enable. 0 PCI_CLK_OUT2 signal is disabled (drive constant ‘0’). 1 PCI_CLK_OUT2 signal is enabled to toggle.
3	PCICOE3	PCI_CLK_OUT3 enable. 0 PCI_CLK_OUT3 signal is disabled (drive constant ‘0’). 1 PCI_CLK_OUT3 signal is enabled to toggle.
4	PCICOE4	PCI_CLK_OUT4 enable. 0 PCI_CLK_OUT4 signal is disabled (drive constant ‘0’). 1 PCI_CLK_OUT4 signal is enabled to toggle.
5	PCICOE5	PCI_CLK_OUT5 enable. 0 PCI_CLK_OUT5 signal is disabled (drive constant ‘0’). 1 PCI_CLK_OUT5 signal is enabled to toggle.
6	PCICOE6	PCI_CLK_OUT6 enable. 0 PCI_CLK_OUT6 signal is disabled (drive constant ‘0’). 1 PCI_CLK_OUT6 signal is enabled to toggle.
7	PCICOE7	PCI_CLK_OUT7 enable. 0 PCI_CLK_OUT7 signal is disabled (drive constant ‘0’). 1 PCI_CLK_OUT7 signal is enabled to toggle.
8–15	PCICD*n*	PCI_CLK_OUT*n* clock division factor. Select if the PCI output clock has the same or half the frequency of CLKIN. 0 PCI_CLK_OUT*n*:CLKIN = 1:1. 1 PCI_CLK_OUT*n*:CLKIN = 1:2. This bit must be cleared if CFG_CLKIN_DIV reset configuration input signal is negated (0) during the power-on reset flow.
16–29	—	Reserved, should be cleared.
30	PCI1CR	PCI1 clock rate. This field select is PCI1 interface operates at CLKIN frequency, or half the frequency of CLKIN. 0 PCI1 interface operates at CLKIN frequency. 1 PCI1 interface operates at half the frequency of CLKIN. This bit must be cleared if CFG_CLKIN_DIV reset configuration input signal is negated (“0”) during the power-on reset flow.
31	PCI2CR	PCI2 clock rate. This field select is PCI2 interface operates at CLKIN frequency, or half the frequency of CLKIN. 0 PCI2 interface operates at CLKIN frequency. 1 PCI2 interface operates at half the frequency of CLKIN. This bit must be cleared if CFG_CLKIN_DIV reset configuration input signal is negated (“0”) during the power-on reset flow.

NOTE

It is the user’s responsibility to set the correct values in this register’s fields, according to CLKIN frequency, CFG_CLKIN_DIV reset configuration input signal value and system requirements. Failing to so will lead to unpredictable behavior.

4.5.2.3 System Clock Control Register (SCCR)

The system clock control register (SCCR), shown in Figure 4-17, controls device units that have a configurable clock ratio.

Address 0x0_0A08 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TSEC1CM		TSEC2CM		—		ENCCM		USBMPHCM		USBDRCM		—			PCICM
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	—															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 4-17. System Clock Control Register (SCCR)

Table 4-36 defines the bit fields of SCCR.

Table 4-36. SCCR Bit Settings

Bits	Name	Description
0–1	TSEC1CM	TSEC1 clock mode. 00 TSEC1 clock is disabled. 01 TSEC1 clock/*csb_clk* ratio is 1:1. 10 TSEC1 clock/*csb_clk* ratio is 1:2 (*csb_clk* has higher frequency than TSEC1). 11 TSEC1 clock/*csb_clk* ratio is 1:3 (*csb_clk* has higher frequency than TSEC1).
2–3	TSEC2CM	TSEC2 and I^2C1 clock mode. 00 TSEC2 and I^2C1 clock is disabled. 01 TSEC2 and I^2C1 clock/*csb_clk* ratio is 1:1. 10 TSEC2 and I^2C1 clock/*csb_clk* ratio is 1:2 (*csb_clk* has higher frequency than TSEC2 and I^2C1). 11 TSEC2 and I^2C1 clock/*csb_clk* ratio is 1:3 (*csb_clk* has higher frequency than TSEC2 and I^2C1). **Note:**
4–5	—	Reserved, should be cleared.
6–7	ENCCM	Encryption core clock mode 00 Encryption core clock is disabled. 01 Encryption core clock/*csb_clk* ratio is 1:1. 10 Encryption core clock/*csb_clk* ratio is 1:2 (*csb_clk* has higher frequency than the encryption core). 11 Encryption core clock/*csb_clk* ratio is 1:3 (*csb_clk* has higher frequency than the encryption core).
8–9	USB MPHCM	USB MPH clock mode 00 USB MPH clock is disabled. 01 USB MPH clock /*csb_clk* ratio is 1:1. 10 USB MPH clock/*csb_clk* ratio is 1:2 (*csb_clk* has higher frequency than the USB MPH). 11 USB MPH clock/*csb_clk* ratio is 1:3 (*csb_clk* has higher frequency than the USB MPH). **Note:** The USB MPH unit must have the same clock ratio as the USB DR unit, unless its clock is disabled.

Table 4-36. SCCR Bit Settings (continued)

Bits	Name	Description
10–11	USB DRCM	USB DR clock mode 00 USB DR clock is disabled. 01 USB DR clock/*csb_clk* ratio is 1:1. 10 USB DR clock/*csb_clk* ratio is 1:2 (*csb_clk* has higher frequency than the USB DR). 11 USB DR clock/*csb_clk* ratio is 1:3 (*csb_clk* has higher frequency than the USB DR). **Note:** The USB DR unit must have the same clock ratio as the USB MPH unit, unless its clock is disabled.
12–14	—	Reserved, should be cleared.
15	PCICM	PCI clock mode. Define the clock mode for all of the PCI complex - PCI1, PCI2 and DMA. 0 PCI complex clocks are disabled. 1 PCI complex clocks are enabled.
16–31	—	Reserved, should be cleared.

4.5.3 Clock Control DDR Registers

The programmable clock control DDR register map occupies 20 bytes of memory-mapped space. Reading undefined portions of the memory map returns all zeros; writing has no effect.

All registers are 32 bits wide located on 32-bit address boundaries. All addresses used in this chapter are offsets from the clock control DDR starting address as defined in Chapter 2, “Memory Map.”

Table 4-37. Clock Control DDR Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
0x00–0x0F	Reserved, should be cleared.	R	0x0000_0000	—
0x10	MCK enable register (MCKENR)	R/W	0xFC00_0000	4.5.3.1/4-41
0x14–0xFF	Reserved, should be cleared.	—	—	—

4.5.3.1 MCK Enable Register (MCKENR)

The MCK clock enable register (MCKENR), shown in Figure 4-18, enables or disables the DDR clock outputs.

Offset 0x10 Access: Read/Write

	0	1	2	3	4	5	6																					31
R W	CE0	CE1	CE2	CE3	CE4	CE5	—																					
Reset	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-18. MCK Clock Register (MCKENR)

Table 4-38 describes MCKENR fields.

Table 4-38. MCKENR Field Description

Bits	Name	Description
0	CE0	Enable/Disable MCK[0] clock output signals 0 Disable MCK[0] and $\overline{\text{MCK}}$[0] 1 Enable MCK[0] and $\overline{\text{MCK}}$[0]
1	CE1	Enable/Disable MCK[1] clock output signals 0 Disable MCK[1] and $\overline{\text{MCK}}$[1] 1 Enable MCK[1] and $\overline{\text{MCK}}$[1]
2	CE2	Enable/Disable MCK[2] clock output signals 0 Disable MCK[2] and $\overline{\text{MCK}}$[2] 1 Enable MCK[2] and $\overline{\text{MCK}}$[2]
3	CE3	Enable/Disable MCK[3] clock output signals 0 Disable MCK[3] and $\overline{\text{MCK}}$[3] 1 Enable MCK[3] and $\overline{\text{MCK}}$[3]
4	CE4	Enable/Disable MCK[4] clock output signals 0 Disable MCK[4] and $\overline{\text{MCK}}$[4] 1 Enable MCK[4] and $\overline{\text{MCK}}$[4]
5	CE5	Enable/Disable MCK[5] clock output signals 0 Disable MCK[5] and $\overline{\text{MCK}}$[5] 1 Enable MCK[5] and $\overline{\text{MCK}}$[5]
6–31	—	Reserved, should be cleared.

Chapter 5
System Configuration

5.1 Introduction

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 5.2, “Local Memory Map Overview and Example”
- Section 5.3, “System Configuration”
- Section 5.4, “Software Watchdog Timer (WDT)”
- Section 5.5, “Real Time Clock Module (RTC)”
- Section 5.6, “Periodic Interval Timer (PIT)”
- Section 5.7, “General-Purpose Timers (GTMs)”
- Section 5.8, “Power Management Control”

5.2 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 nine local access windows. Each of these windows maps a region of memory to a particular target interface, such as the DDR SDRAM controller or the PCI controller. Note that the local access windows do not perform any address translation. The size of each window can be configured from 4 Kbytes to 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	PCI	Both PCI1 and PCI2
6	PCI	Both PCI1 and PCI2
7	DDR SDRAM	—
8	DDR SDRAM	—

Figure 5-1 shows an example memory map.

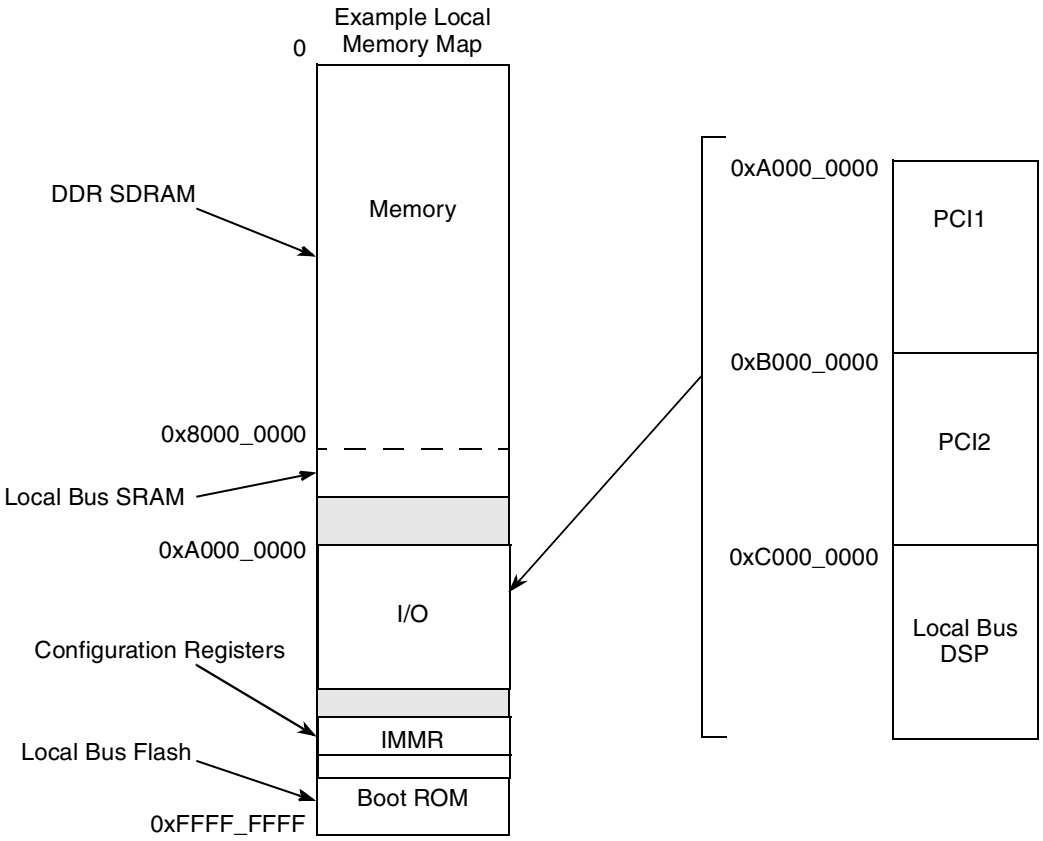


Figure 5-1. Local Memory Map Example

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

Table 5-2. Local Access Windows Example

Window	Base Address	Size	Target Interface
7	0x0000_0000	2 Gbytes	DDR SDRAM
2	0x8000_0000	1 Mbyte	Local bus
5	0xA000_0000	256 Mbytes	PCI1
6	0xB000_0000	256 Mbytes	PCI2
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
4, 8	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 4.3.2.2.4, “Boot ROM Location,”) and Section 5.2.4.3.1, “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 5.2.4.1, “Internal Memory Map Registers Base Address Register (IMMRBAR).”

5.2.1 Address Translation and Mapping

In addition to any address translation performed by the e300c1 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 spaces of PCI1 or PCI2, which may be much larger than the local space.

Inbound windows perform address translation from the external address spaces of PCI1 or PCI2 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[1]

[1] 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.

5.2.2 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 5.2.4.1, “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.

5.2.3 Local Access Windows

As demonstrated in the address map overview in Section 5.2, “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 inbound windows.

The local access window registers exist as part of the local access block in the system configuration registers. See Section 5.3.2, “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.

5.2.3.1 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 Memory Offset (Hex)	Register	Access	Reset	Section/Page
0x0_0000	Internal memory map base address register (IMMRBAR)	R/W	0xFF40_0000	5.2.4.1/5-5
0x0_0004	Reserved	—	—	—
0x0_0008	Alternate configuration base address register (ALTCBAR)	R/W	0x0000_0000	5.2.4.2/5-7
0x0_000C–0x0_001C	Reserved	—	—	—
0x0_0020	LBC local access window 0 base address register (LBLAWBAR0)	R/W	0x0000_0000[1]	5.2.4.3/5-7
0x0_0024	LBC local access window 0 attribute register (LBLAWAR0)	R/W	0x0000_0000[2]	5.2.4.4/5-8
0x0_0028	LBC local access window 1 base address register (LBLAWBAR1)	R/W	0x0000_0000	5.2.4.3/5-7
0x0_002C	LBC local access window 1 attribute register (LBLAWAR1)	R/W	0x0000_0000	5.2.4.4/5-8
0x0_0030	LBC local access window 2 base address register (LBLAWBAR2)	R/W	0x0000_0000	5.2.4.3/5-7
0x0_0034	LBC local access window 2 attribute register (LBLAWAR2)	R/W	0x0000_0000	5.2.4.4/5-8
0x0_0038	LBC local access window 3 base address register (LBLAWBAR3)	R/W	0x0000_0000	5.2.4.3/5-7

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

Local Memory Offset (Hex)	Register	Access	Reset	Section/Page
0x0_003C	LBC local access window 3 attribute register (LBLAWAR3)	R/W	0x0000_0000	5.2.4.4/5-8
0x0_0040–0x0_005C	Reserved	—	—	—
0x0_0060	PCI local access window0 base address register (PCILAWBAR0)	R/W	0x0000_0000[3]	5.2.4.5/5-9
0x0_0064	PCI local access window0 attribute register (PCILAWAR0)	R/W	0x0000_0000[4]	5.2.4.6/5-10
0x0_0068	PCI local access window1 base address register (PCILAWBAR1)	R/W	0x0000_0000[5]	5.2.4.5/5-9
0x0_006C	PCI local access window1 attribute register (PCILAWAR1)	R/W	0x0000_0000	5.2.4.6/5-10
0x0_0070–0x0_009C	Reserved	—	—	—
0x0_00A0	DDR local access window0 base address register (DDRLAWBAR0)	R/W	0x0000_0000[6]	5.2.4.7/5-12
0x0_00A4	DDR local access window0 attribute register (DDRLAWAR0)	R/W	0x0000_0000[7]	5.2.4.8/5-13
0x0_00A8	DDR local access window1 base address register (DDRLAWBAR1)	R/W	0x0000_0000	5.2.4.7/5-12
0x0_00AC	DDR local access window1 attribute register (DDRLAWAR1)	R/W	0x0000_0000	5.2.4.8/5-13
0x0_00B0–0x0_00FC	Reserved	—	—	—

1 Depends on reset configuration word high values. See Section 5.2.4.3.1, “LBLAWBAR0[BASE_ADDR] Reset Value,” for details.

2 Depends on reset configuration word high values. See Section 5.2.4.4.1, “LBLAWAR0[EN] and LBLAWAR0[SIZE] Reset Value,” for details.

3 Depends on reset configuration word high values. See Section 5.2.4.5.1, “PCILAWBAR0[BASE_ADDR] Reset Value” for details.

4 Depends on reset configuration word high values. See Section 5.2.4.7.1, “DDRLAWBAR0[BASE_ADDR] Reset Value,” for details.

5 Depends on reset configuration word high values. See Section 5.2.4.6.1, “PCILAWAR0[EN] and PCILAWAR0[SIZE] Reset Value,” for details.

6 Depends on reset configuration word high values. See Section 5.2.4.7.1, “DDRLAWBAR0[BASE_ADDR] Reset Value,” for details.

7 Depends on reset configuration word high values. See Section 5.2.4.8.1, “DDRLAWAR0[EN] and DDRLAWAR0[SIZE] Reset Value,” for details.

5.2.4 Local Access Register Descriptions

5.2.4.1 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, IMMRBAR always points to itself.

5.2.4.1.1 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 an external host on PCI is configuring the device, it should set IMMRBAR to the desired final location before the e300c1 core is released to boot.
  - 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.

5.2.4.2 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," of Chapter 17, "I^2C Interfaces," 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.

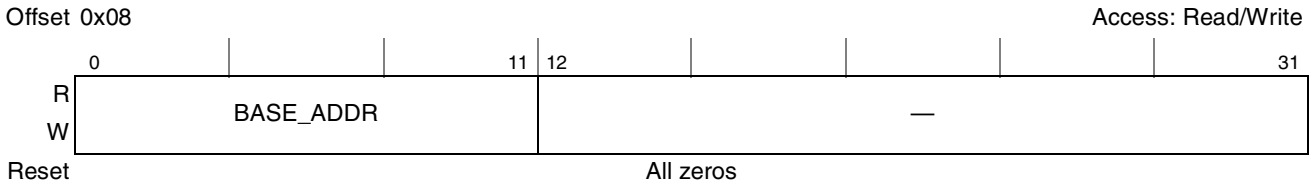


Figure 5-3. Alternate Configuration Base Address Register (ALTCBAR)

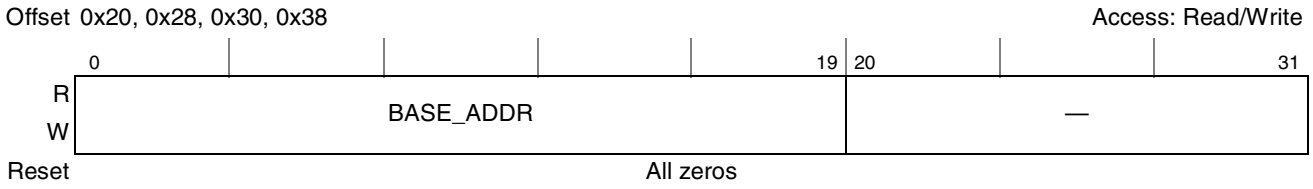
Table 5-6 defines the bit fields of ALTCBAR.

Table 5-6. ALTCBAR Bit Settings

Bits	Name	Description
0–11	BASE_ADDR	Identifies the12 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.

5.2.4.3 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.



1. The LBLAWBAR0[BASE_ADDR] reset value depends on the reset configuration word high values. See Section 5.2.4.3.1, "LBLAWBAR0[BASE_ADDR] Reset Value," for a detailed description.

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

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 n. 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.

5.2.4.3.1 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.

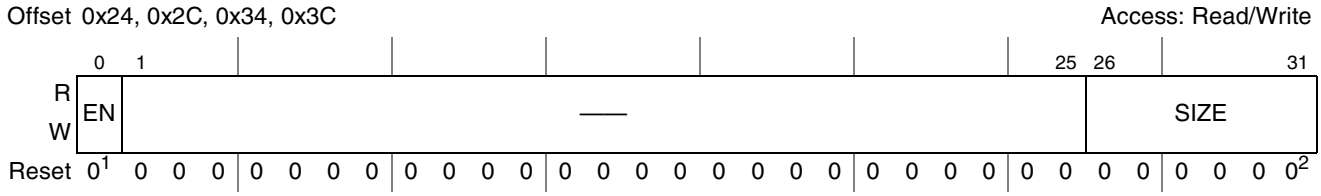
Table 5-8 defines the reset value of LBLAWBAR0[BASE_ADDR].

Table 5-8. LBLAWBAR0[BASE_ADDR] Reset Value

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

5.2.4.4 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.



1 The LBLAWAR0[EN] reset value depends on the reset configuration word high values. See Section 5.2.4.4.1, "LBLAWAR0[EN] and LBLAWAR0[SIZE] Reset Value," for a detailed description.

2 The LBLAWAR0[SIZE] reset value is always 0b010110, meaning an 8-Mbyte local access window. See Section 5.2.4.4.1, "LBLAWAR0[EN] and LBLAWAR0[SIZE] Reset Value," for a detailed description.

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

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 n is disabled. 1 Local bus local access window n 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)

Bits	Name	Description
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.

5.2.4.4.1 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.

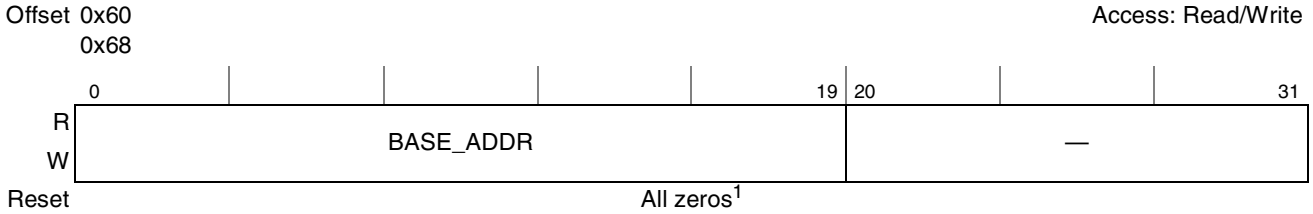
Table 5-10 defines the reset value for LBLAWAR0[EN].

Table 5-10. LBLAWAR0[EN] Reset Value

RCWHR[ROMLOC]	LBLAWAR0[EN] Reset Value	Description
000–100	0	e300c1 core boot not performed from a local bus device.
101–111	1	e300c1 core boot performed from a local bus device. Local bus 8-Mbyte ($2^{(22+1)}$) local access window is enabled.

5.2.4.5 PCI Local Access Window *n* Base Address Register (PCILAWBAR0–PCILAWBAR1)

The PCI local access window *n* base address registers (PCILAWBAR0–PCILAWBAR1) are shown in Figure 5-6.



1 The reset value of PCILAWBAR0[BASE_ADDR] depends on the reset configuration word high values. See Section 5.2.4.5.1, "PCILAWBAR0[BASE_ADDR] Reset Value," for a detailed description.

Figure 5-6. PCI Local Access Window *n* Base Address Registers (PCILAWBAR0–PCILAWBAR1)

Table 5-11 defines the bit fields of PCILAWBAR0–PCILAWBAR1.

Table 5-11. PCILAWBAR0–PCILAWBAR1 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 PCILAWARn[SIZE].
20–31	—	Reserved. Write has no effect, read returns 0.

5.2.4.5.1 PCILAWBAR0[BASE_ADDR] Reset Value

The core may use a PCI peripheral device to fetch its boot vector. For this purpose, the PCILAWBAR0[BASE_ADDR] reset value is set according to the value set in the reset configuration word high BMS field.

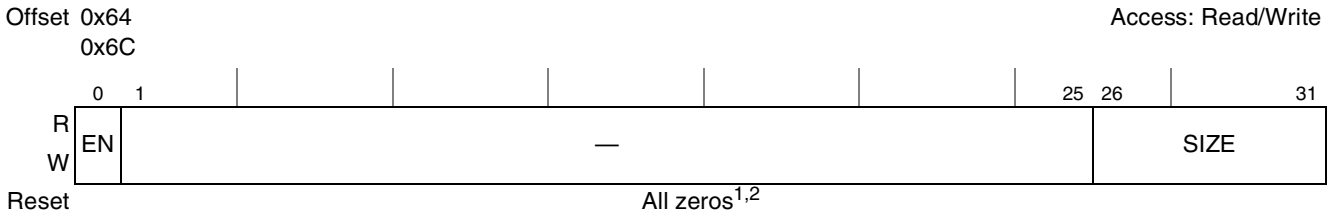
Table 5-12 defines the reset value of PCILAWBAR0[BASE_ADDR].

Table 5-12. PCILAWBAR0[BASE_ADDR] Reset Value

RCWHR[BMS]	PCILAWBAR0[BASE_ADDR] Reset Value
0	0x00000
1	0xFF800

5.2.4.6 PCI Local Access Window *n* Attributes Registers (PCILAWAR0–PCILAWAR1)

The PCI local access window *n* attributes registers (PCILAWAR0–PCILAWAR1) are shown in Figure 5-7.



1 The reset value of PCILAWAR0[EN] depends on the reset configuration word high values. See Section 5.2.4.6.1, “PCILAWAR0[EN] and PCILAWAR0[SIZE] Reset Value,” for a detailed description.

2 The reset value of PCILAWAR0[SIZE] is always 0b010110, meaning an 8-Mbyte local access window. See Section 5.2.4.6.1, “PCILAWAR0[EN] and PCILAWAR0[SIZE] Reset Value,” for a detailed description

Figure 5-7. PCI Local Access Window *n* Attributes Registers (PCILAWAR0–PCILAWAR1)

Table 5-13 defines the bit fields of PCILAWAR0–PCILAWAR1.

Table 5-13. PCILAWAR0–PCILAWAR1 Bit Settings

Bits	Name	Description
0	EN	0 The PCI local access window *n* is disabled. 1 The PCI local access window *n* is enabled and other PCILAWAR*n* and PCILAWBAR*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.

5.2.4.6.1 PCILAWAR0[EN] and PCILAWAR0[SIZE] Reset Value

The core may use a PCI peripheral device to fetch its boot vector. For this purpose an 8-Mbyte ($2^{(22+1)}$) local access window is defined by the PCILAWBAR0[SIZE] reset value, and PCILAWAR0 is enabled according to the value set in the reset configuration word high ROMLOC field.

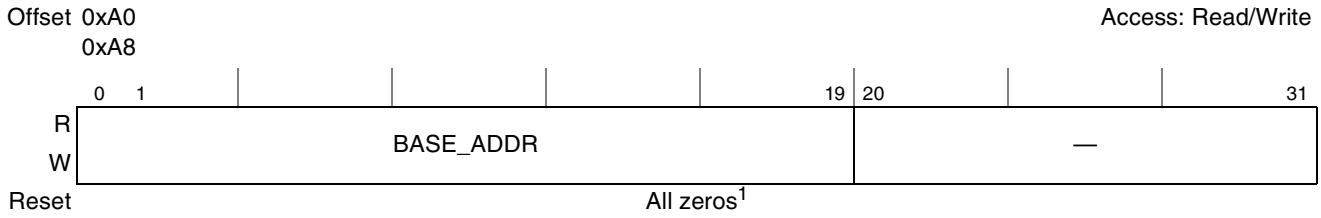
Table 5-14 defines the reset value of PCILAWAR0[EN].

Table 5-14. PCILAWAR0[EN] Reset Value

RCWHR[ROMLOC]	PCILAWR0[EN] Reset Value	Description
000, 011–111	0	e300c1 core boot not performed from a PCI device.
001	1	e300c1 core boot performed from a PCI1 device. PCI 8-Mbyte ($2^{(22+1)}$) local access window is enabled.
010	1	e300c1 core boot performed from a PCI2 device. PCI 8-Mbyte ($2^{(22+1)}$) local access window is enabled.

5.2.4.7 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.



1 The reset value of DDRLAWBAR0[BASE_ADDR] depends on the reset configuration word high values. See Section 5.2.4.7.1, "DDRLAWBAR0[BASE_ADDR] Reset Value," for a detailed description

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

Table 5-15 defines the bit fields of DDRLAWBAR0–DDRLAWBAR1.

Table 5-15. 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.

5.2.4.7.1 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.

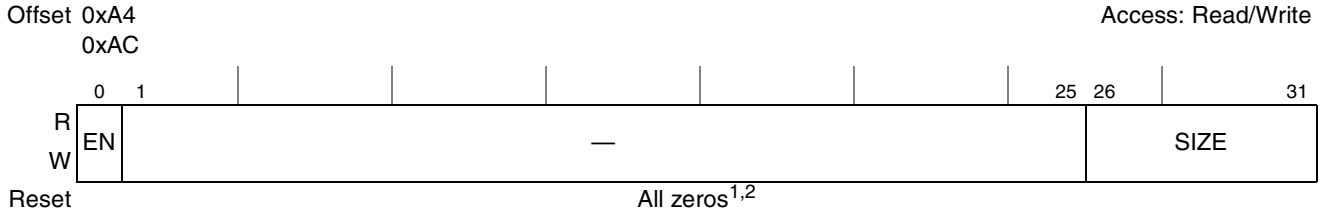
Table 5-16 defines the reset value DDRLAWBAR0.

Table 5-16. DDRLAWBAR0[BASE_ADDR] Reset Value

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

5.2.4.8 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.



1 The reset value of DDRLAWAR0[EN] depends on the reset configuration word high values. See Section 5.2.4.8.1, "DDRLAWAR0[EN] and DDRLAWAR0[SIZE] Reset Value," for a detailed description.

2 The reset value of DDRLAWAR0[SIZE] is always 0b010110, meaning an 8-Mbyte local access window. See Section 5.2.4.8.1, "DDRLAWAR0[EN] and DDRLAWAR0[SIZE] Reset Value," for a detailed description.

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

Table 5-17 defines the bit fields of DDRLAWAR0–DDRLAWAR1.

Table 5-17. 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.

5.2.4.8.1 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 5-18 defines the reset value DDRLAWAR0[EN] and DDRLAWAR0[SIZE].

Table 5-18. DDRLAWAR0[EN] Reset Value

RCWHR[ROMLOC]	DDRLAWAR0[EN] Reset Value	Description
000	1	e300c1 core boot performed from a DDR SDRAM device. DDR 8-Mbyte ($2^{(22+1)}$) local access window is enabled.
else	0	e300c1 core boot not performed from a DDR SDRAM device.

5.2.5 Precedence of Local Access Windows

If two local access windows overlap, the lower numbered window takes precedence (see Table 5-1for window numbers). For instance, if two windows are set up as shown in Table 5-19, 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 5-19. Overlapping Local Access Windows

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

5.2.6 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 e300c1 core, the read of LBLAWAR3 should be followed by an **isync** instruction.

5.2.7 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 PCI1 and PCI2 interfaces have 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.

5.2.8 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, both the PCI1 and the PCI2 blocks have outbound address translation units.

The PCI1/PCI2 controllers have six outbound windows plus a default window. See Section 11.3, “Memory Map/Register Definition,” for a detailed description of the PCI1/PCI2 outbound windows.

5.2.9 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 PCI1 or PCI2 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. Both the PCI1 controllerand the PCI2 controller have inbound address translation units.

5.2.9.1 PCI1/PCI2 Inbound Windows

The PCI1/PCI2 controllers have six general inbound windows plus a dedicated window for memory mapped configuration accesses (PIMMR). These windows have a one-to-one correspondence with the base address registers in the PCI1/PCI2 programming model. Updating one automatically updates the other. There is no default inbound window; if a PCI1/PCI2 address does not match one of the inbound windows, this processor does not respond with an assertion of $\overline{\text{PCI_DEVSEL}}$. See Section 13.4.6, “PCI Inbound Address Translation,” for a detailed description of the PCI1/PCI2 inbound windows.

5.2.10 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 5.2.4.1, “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.

5.2.11 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.

The PCI1/PCI2 base address for accessing the local IMMR memory is selectable through the PCI internal memory map register (PIMMR), at offset 0x10, described in Section 13.3.2.13, "PCI Inbound Base Address Registers (PIBARn)." When the device is a PCI agent, an external PCI master sets this register by running a PCI configuration cycle. Subsequent memory accesses by a PCI master to the PCI address range indicated by PIMMR are translated to the local address indicated by the current setting of IMMRBAR.

5.3 System Configuration

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

5.3.1 System Configuration Register Memory Map

Table 5-20 shows the memory map for the system configuration registers.

Table 5-20. System Configuration Register Memory Map

Local Memory Offset (Hex)	Register	Access	Reset	Section/Page
0x00100	System general purpose register low (SGPRL)	R/W	0x0000_0000	5.3.2.1/5-17
0x00104	System general purpose register high (SGPRH)	R/W	0x0000_0000	5.3.2.2/5-17
0x00108	System part and revision ID register (SPRIDR)	R	0x8030_0100	5.3.2.3/5-18
0x0010C	Reserved	—	—	—
0x00110	System priority configuration register (SPCR)	R/W	0x0000_0000	5.3.2.4/5-19
0x00114	System I/O configuration register low (SICRL)	R/W	0x0000_0000[1]	5.3.2.5/5-21
0x00118	System I/O configuration register high (SICRH)	R/W	0x0000_0000	5.3.2.6/5-24
0x0011C–0x00124	Reserved	—	—	—
0x00128	DDR control driver register (DDRCDR)	R/W	0x7304_0001	5.3.2.8/5-28
0x0012C	DDR debug status register (DDRDSR)	R	0x3300_0000	5.3.2.9/5-30
0x00130–0x001FC	Reserved	—	—	—

[1] Depends on the reset configuration word high configuration values.

5.3.2 System Configuration Registers

5.3.2.1 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.

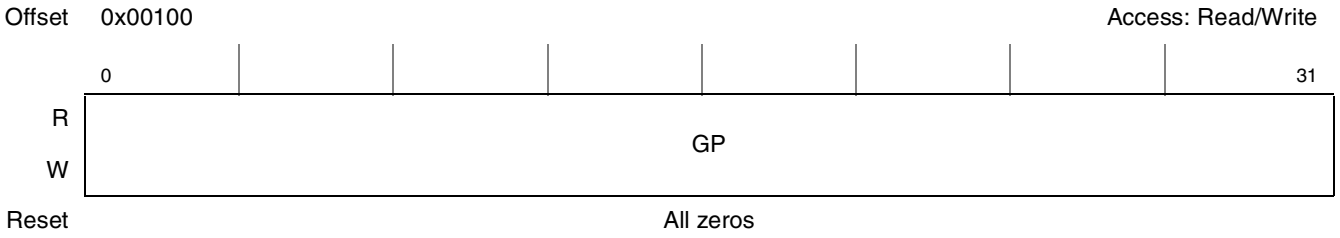


Figure 5-10. System General Purpose Register Low (SGPRL)

Table 5-21 defines the bit fields of SGPRL.

Table 5-21. SGPRL Bit Settings

Bits	Name	Description
0–31	GP	General purpose

5.3.2.2 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.

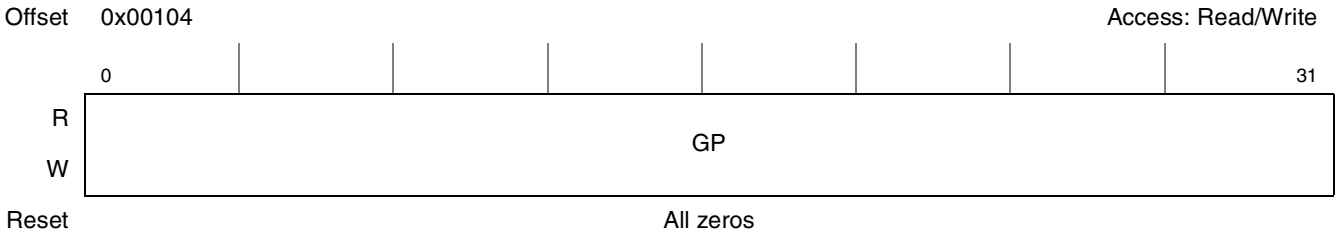


Figure 5-11. System General Purpose Register High (SGPRH)

Table 5-22 defines the bit fields of SGPRH.

Table 5-22. SGPRH Bit Settings

Bits	Name	Description
0–31	GP	General purpose

5.3.2.3 System Part and Revision ID Register (SPRIDR)

SPRIDR, shown in Figure 5-12, provides information about the device and revision numbers.

Offset 0x00108 Access: Read only

	0	15	16	31
R	PARTID		REVID	
W				
Reset	See Table 5-24 and Table 5-25 for reset values of this register.			

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

Table 5-23 defines the bit fields of SPRIDR.

Table 5-23. SPRIDR Bit Settings

Bits	Name	Description
0–15	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 Table 5-24 for values of this field.
16–31	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 Table 5-25 for values of this field.

5.3.2.3.1 SPRIDR[PARTID] Coding

Table 5-24 defines the reset values of SPRIDR[PARTID].

Table 5-24. PARTID Coding

PARTID	Device Name	Package Type
0x8030	MPC8349EA	TBGA
0x8031	MPC8349A	TBGA
0x8032	MPC8347EA	TBGA
0x8033	MPC8347A	TBGA
0x8034	MPC8347EA	PBGA
0x8035	MPC8347A	PBGA
0x8036	MPC8343EA	PBGA
0x8037	MPC8343A	PBGA

Table 5-25 defines the reset values of SPRIDR[REVID].

Table 5-25. REVID Coding

REVID	Device Revision
0x0100	1.0
0x0101	1.1
0x0300	3.0

5.3.2.4 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.

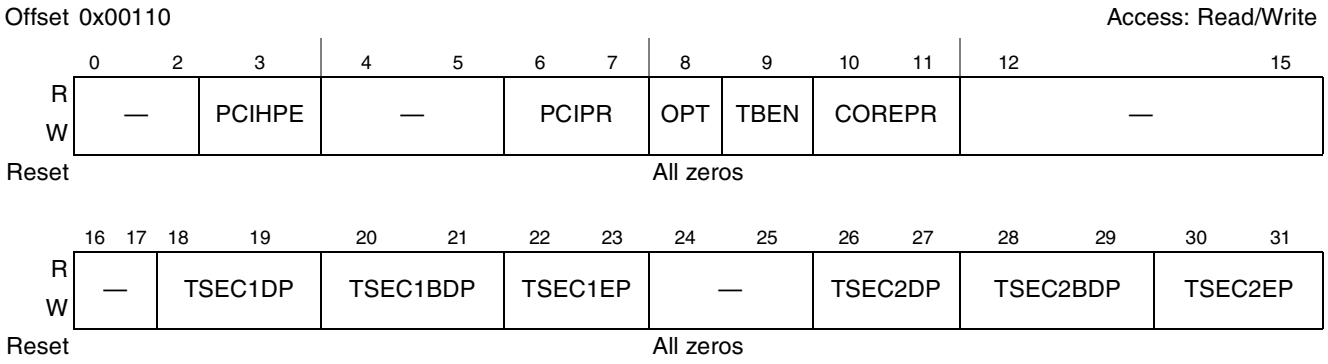


Figure 5-13. System Priority Configuration Register (SPCR)

Table 5-26 defines the bit fields of SPCR.

Table 5-26. SPCR Bit Settings

Bits	Name	Description
0–2	—	Reserved. Should be cleared.
3	PCIHPE	PCI highest priority enable. If this bit is set, the PCI bridge is permitted to request the coherent system bus (CSB) with highest priority, regardless of SPCR[PCIPR] value, when it needs to complete a posted write transaction from an external PCI master. To follow PCI ordering rules specifications, the PCI bridge must flush any outstanding write transactions before it can start a new read transaction. Setting this bit allows faster flushing of the outstanding write transactions coming from the PCI bus onto the CSB and to the device targets, such as DDR SDRAM and local bus memories.
4–5	—	Reserved. Should be cleared.
6–7	PCIPR	PCI bridge CSB request priority. The level of priority can be chosen from 4 possible levels. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority)

Table 5-26. SPCR Bit Settings (continued)

Bits	Name	Description
8	OPT	Optimize. Setting this bit may enhance the performance of transactions issued to the internal coherent system bus (CSB) by the security engine (SEC) and the 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	e300c1 core time base unit enable 0 Time base unit is disabled. 1 Time base unit is enabled.
10–11	COREPR	e300c1 core CSB request priority. The priority level for the core in accessing the CSB can be chosen from 4 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	TSEC1DP	TSEC1 data priority. Selects the CSB request priority driven by TSEC1 when it requires to transfer data on this bus. The level of priority can be chosen from 4 possible levels. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority)
20–21	TSEC1BDP	TSEC1 buffer descriptor priority. Selects the CSB request priority driven by TSEC1 when it requires to transfer a buffer descriptor (BD) on this bus. The level of priority can be chosen from 4 possible levels. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority)
22–23	TSEC1EP	TSEC1 emergency priority. Selects the CSB request priority driven by TSEC1 when an emergency condition occurs. The level of priority can be chosen from 4 possible levels. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority)
24–25	—	Reserved, should be cleared.
26–27	TSEC2DP	TSEC2 data priority. Selects the CSB request priority driven by TSEC2 when it requires to transfer data on this bus. The level of priority can be chosen from 4 possible levels. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority)

Table 5-26. SPCR Bit Settings (continued)

Bits	Name	Description
28–29	TSEC2BDP	TSEC2 buffer descriptor priority. Selects the CSB request priority driven by TSEC2 when it requires to transfer a buffer descriptor (BD) on this bus. The level of priority can be chosen from 4 possible levels. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority)
30–31	TSEC2EP	TSEC2 emergency priority. Selects the CSB request priority driven by TSEC2 when an emergency condition occurs. The level of priority can be chosen from 4 possible levels. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority)

5.3.2.5 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.Figure 5-14 shows SICRL.

Offset 0x00114 Access: Read/Write

	0	1	2	3	4	5	6	7
R W	LDP_A	USB port_1	USB port_0	SPI	UART		GPIO1_A	GPIO1_B
Reset[1]	1	0	0	0	0	0	0	0

	8	9	10	11	12	13	14	15
R W	GPIO1_C	GPIO1_D	GPIO1_E	GPIO1_F		GPIO1_G	GPIO1_H	GPIO1_I
Reset	All zeros							

	16	17	18	19	20	21	22	23
R W	GPIO1_J	GPIO1_K	GPIO1_L		DMA_A	DMA_B	DMA_C	DMA_D
Reset	All zeros							

	24	25	26	27	28	29	30	31
R W	DMA_E	DMA_F	DMA_G	DMA_H	DMA_I	DMA_J	—	
Reset	All zeros							

[1] The reset value of SICRL[0] depends on the value of RCWH[30] which is loaded into the reset configuration word.

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

Table 5-27 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 '0b*N*' or '0b*NN*' according to whether a group has one or two control bits, respectively.

Table 5-27. SICRL Bit Settings

SICRL[Bits] Value		0b0/0b00	0b1/0b01	0b10	0b11
Bits	**Group**	**Pin Function 0**	**Pin Function 1**	**Pin Function 2**	**Pin Function 3**
0	LDP_A	$\overline{\text{LCS[5]}}$	LDP[3]	—	—
		$\overline{\text{LCS[4]}}$	LDP[2]	—	—
		$\overline{\text{CKSTOP_IN}}$	LDP[1]	—	—
		$\overline{\text{CKSTOP_OUT}}$	LDP[0]	—	—
1	USB port 1	MPH1_D0_ENABLEN	DR_D0_ENABLEN	—	—
		MPH1_D1_SER_TXD	DR_D1_SER_TXD	—	—
		MPH1_D2_VMO_SE0	DR_D2_VMO_SE0	—	—
		MPH1_D3_SPEED	DR_D3_SPEED	—	—
		MPH1_D4_DP	DR_D4_DP	—	—
		MPH1_D5_DM	DR_D5_DM	—	—
		MPH1_D6_SER_RCV	DR_D6_SER_RCV	—	—
		MPH1_D7_DRVVBUS	DR_D7_DRVVBUS	—	—
		MPH1_NXT	DR_SESS_VLD_NXT	—	—
		MPH1_DIR_DPPULLUP	DR_DIR_XCVR_SEL_DPPULLUP	—	—
		MPH1_STP_SUSPEND	DR_STP_SUSPEND	—	—
		MPH1_PWRFAULT	DR_RX_ERROR_PWRFAULT	—	—
		MPH1_PCTL0	DR_TX_VALID_PCTL0	—	—
		MPH1_PCTL1	DR_TX_VALIDH_PCTL1	—	—
		MPH1_CLK	DR_CLK	—	—

Table 5-27. SICRL Bit Settings (continued)

SICRL[Bits] Value		0b0/0b00	0b1/0b01	0b10	0b11
Bits	**Group**	**Pin Function 0**	**Pin Function 1**	**Pin Function 2**	**Pin Function 3**
2	USB port 0	MPH0_D0_ENABLEN	DR_D8_CHGVBUS	—	—
		MPH0_D1_SER_TXD	DR_D9_DCHGVBUS	—	—
		MPH0_D2_VMO_SE0	DR_D10_DPPD	—	—
		MPH0_D3_SPEED	DR_D11_DMMD	—	—
		MPH0_D4_DP	DR_D12_VBUS_VLD	—	—
		MPH0_D5_DM	DR_D13_SESS_END	—	—
		MPH0_D6_SER_RCV	DR_D14	—	—
		MPH0_D7_DRVVBUS	DR_D15_IDPULLUP	—	—
		MPH0_NXT	DR_RX_ACTIVE_ID	—	—
		MPH0_DIR_DPPULLUP	DR_RESET	—	—
		MPH0_STP_SUSPEND	DR_TX_READY	—	—
		MPH0_PWRFAULT	DR_RX_VALIDH	—	—
		MPH0_PCTL0	DR_LINE_STATE0	—	—
		MPH0_PCTL1	DR_LINE_STATE1	—	—
		MPH0_CLK	DR_RX_VALID	—	—
3	SPI	SPIMOSI	$\overline{\text{LCS}}$[6]	—	—
		SPIMISO	$\overline{\text{LCS}}$[7]	—	—
4–5	UART	UART_SIN1	MSRCID[2]	LSRCID[2]	—
		UART_SIN2	MSRCID[3]	LSRCID[3]	—
		UART_SOUT1	MSRCID[0]	LSRCID[0]	—
		UART_SOUT2	MSRCID[1]	LSRCID[1]	—
		$\overline{\text{UART_CTS}}$[1]	MSRCID[4]	LSRCID[4]	—
		$\overline{\text{UART_CTS}}$[2]	MDVAL	LDVAL	—
6[1]	GPIO1_A	GPIO1[0]	GTM1_TIN1/GTM2_TIN2	—	—
7[1]	GPIO1_B	GPIO1[1]	$\overline{\text{GTM1_TGATE1}}$/$\overline{\text{GTM2_TGATE2}}$	—	—
8[1]	GPIO1_C	GPIO1[2]	$\overline{\text{GTM1_TOUT1}}$	—	—
9[1]	GPIO1_D	GPIO1[3]	GTM1_TIN2/GTM2_TIN1	—	—
10[1]	GPIO1_E	GPIO1[4]	$\overline{\text{GTM1_TGATE2}}$/$\overline{\text{GTM2_TGATE1}}$	—	—
11–12	GPIO1_F	GPIO1[5]	$\overline{\text{GTM1_TOUT2}}$	$\overline{\text{GTM2_TOUT1}}$	$\overline{\text{DMA_DDONE1}}$
13[2]	GPIO1_G	GPIO1[6]	GTM1_TIN3/GTM2_TIN4	—	—
14[2]	GPIO1_H	GPIO1[7]	$\overline{\text{GTM1_TGATE3}}$/$\overline{\text{GTM2_TGATE4}}$	—	—

Table 5-27. SICRL Bit Settings (continued)

SICRL[Bits] Value		0b0/0b00	0b1/0b01	0b10	0b11
Bits	**Group**	**Pin Function 0**	**Pin Function 1**	**Pin Function 2**	**Pin Function 3**
15[2]	GPIO1_I	GPIO1[8]	$\overline{\text{GTM1_TOUT3}}$	—	—
16[2]	GPIO1_J	GPIO1[9]	GTM1_TIN4/GTM2_TIN3	—	—
17[2]	GPIO1_K	GPIO1[10]	$\overline{\text{GTM1_TGATE4}}$/$\overline{\text{GTM2_TGATE3}}$	—	—
18–19	GPIO1_L	GPIO1[11]	$\overline{\text{GTM1_TOUT4}}$	$\overline{\text{GTM2_TOUT3}}$	$\overline{\text{DMA_DDONE3}}$
20	DMA_A	Controlled by SICRL[6]	$\overline{\text{DMA_DREQ0}}$	—	—
21	DMA_B	Controlled by SICRL[7]	$\overline{\text{DMA_DACK0}}$	—	—
22	DMA_C	Controlled by SICRL[8]	$\overline{\text{DMA_DDONE0}}$	—	—
23	DMA_D	Controlled by SICRL[9]	$\overline{\text{DMA_DREQ1}}$	—	—
24	DMA_E	Controlled by SICRL[10]	$\overline{\text{DMA_DACK1}}$	—	—
25	DMA_F	Controlled by SICRL[13]	$\overline{\text{DMA_DREQ2}}$	—	—
26	DMA_G	Controlled by SICRL[14]	$\overline{\text{DMA_DACK2}}$	—	—
27	DMA_H	Controlled by SICRL[15]	$\overline{\text{DMA_DDONE2}}$	—	—
28	DMA_I	Controlled by SICRL[16]	$\overline{\text{DMA_DREQ3}}$	—	—
29	DMA_J	Controlled by SICRL[17]	$\overline{\text{DMA_DACK3}}$	—	—
20–31	Reserved	—	—	—	—

1 The setting of the signals controlled by bits 6–10 also depends on bits 20–24.

2 The setting of the signals controlled by bits 13–17 also depends on bits 25–29.

5.3.2.6 System I/O Configuration Register High (SICRH)

The system I/O configuration register high, 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 select which function is used by a certain group of the device pins. The reset value of this register depends on TSEC1M and TSEC2M fields setting in the reset configuration word high. This is used to avoid contention in systems not using TBI or RTBI modes. In these systems, the device pins are not driven during and after reset. The function of these pins can be changed by writing to this register during system initialization. The reset value of TSOBI1 and TSOBI2 also depends on the TSEC1M and TSEC2M fields settings in the reset configuration word high in order to select the correct output buffer impedance for full or reduced TSEC pin mode.

Offset 0x00118 Access: Read/Write

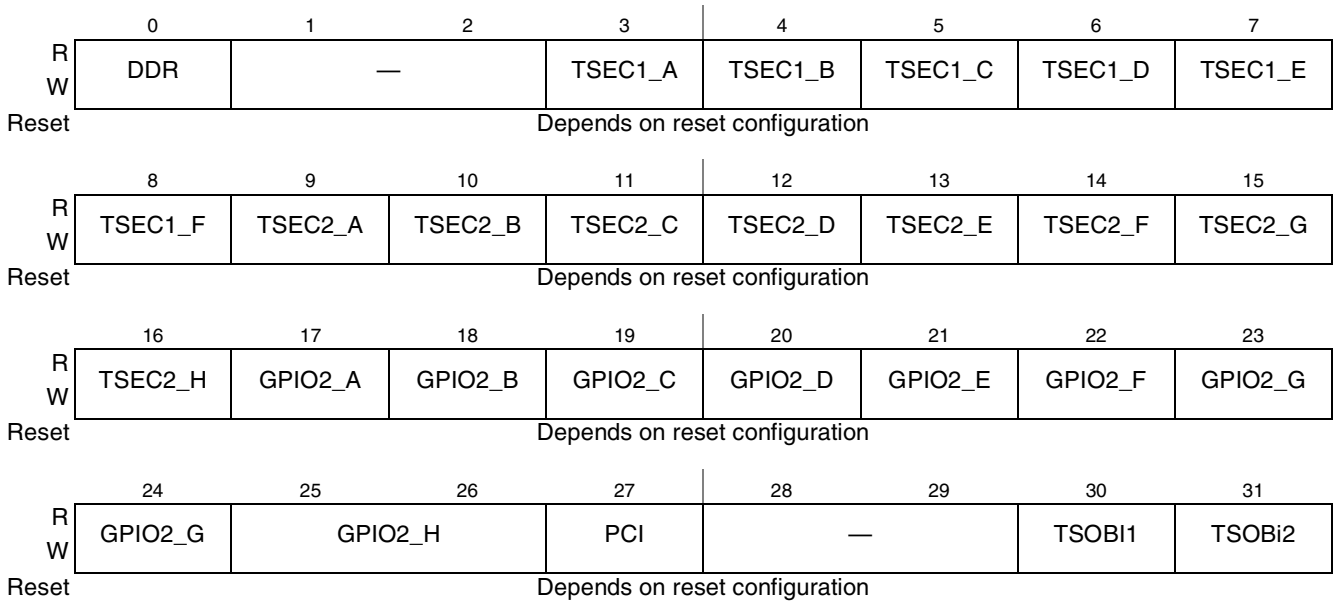


Figure 5-15. System I/O Configuration Register High (SICRH)

Table 5-28 defines the bit fields of SICRH. 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 have the value of 0b0 or 0b1 only. Other groups may have three options (shown as Pin Function 0, Pin Function 1, and Pin Function 2), and therefore two control bits. In this case they can have the value of 0b00, 0b01 or 0b10. A value of 0b11 is illegal for all groups. Use the 0b*N* or 0b*NN* notations according to a group having one or two control bits respectively.

Note that bits 30 and 31 (TSOBI1 and TSOBI2), which control TSEC output buffer impedance, are described in Table 5-29.

Table 5-28. SICRH Bit Settings

SICRH[Bits] Value:		0b0/0b00	0b1/0b01	0b10	Reset Value
Bits	**Group**	**Pin Function 0**	**Pin Function 1**	**Pin Function 2**	
0	DDR	MECC[0]	MSRCID[0]	—	0
		MECC[1]	MSRCID[1]	—	
		MECC[2]	MSRCID[2]	—	
		MECC[3]	MSRCID[3]	—	
		MECC[4]	MSRCID[4]	—	
		MECC[5]	MDVAL	—	
1–2	Reserved	—	—	—	00
3	TSEC1_A	TSEC1_COL	GPIO2[20]	—	1
4	TSEC1_B	TSEC1_CRS	GPIO2[21]	—	0 TBI, RTBI 1 Else

Table 5-28. SICRH Bit Settings (continued)

SICRH[Bits] Value:		0b0/0b00	0b1/0b01	0b10	Reset Value
Bits	Group	Pin Function 0	Pin Function 1	Pin Function 2	
5	TSEC1_C	TSEC1_RXD[7]	GPIO2[22]	—	0 TBI 1 RTBI, Else
		TSEC1_RXD[6]	GPIO2[23]	—	
		TSEC1_RXD[5]	GPIO2[24]	—	
		TSEC1_RXD[4]	GPIO2[25]	—	
6	TSEC1_D	TSEC1_RX_ER	GPIO2[26]	—	0 TBI 1 RTBI, Else
7	TSEC1_E	TSEC1_TXD[7]	GPIO2[27]	—	0 TBI 1 RTBI, Else
		TSEC1_TXD[6]	GPIO2[28]	—	
		TSEC1_TXD[5]	GPIO2[29]	—	
		TSEC1_TXD[4]	GPIO2[30]	—	
8	TSEC1_F	TSEC1_TX_ER	GPIO2[31]	—	0 TBI 1 RTBI, Else
9	TSEC2_A	TSEC2_COL	GPIO1[21]	—	1
10	TSEC2_B	TSEC2_CRS	GPIO1[22]	—	0 TBI, RTBI 1 Else
11	TSEC2_C	TSEC2_RXD3	GPIO1[13]	—	0 TBI, RTBI 1 Else
		TSEC2_RXD2	GPIO1[14]	—	
		TSEC2_RXD1	GPIO1[15]	—	
		TSEC2_RXD0	GPIO1[16]	—	
		TSEC2_TXD3	GPIO1[17]	—	
		TSEC2_TXD2	GPIO1[18]	—	
		TSEC2_TXD1	GPIO1[19]	—	
		TSEC2_TXD0	GPIO1[20]	—	
		TSEC2_RX_DV	GPIO1[23]	—	
		TSEC2_TX_EN	GPIO1[12]	—	
12	TSEC2_D	TSEC2_RXD[7]	GPIO1[26]	—	0 TBI 1 RTBI, Else
		TSEC2_RXD[6]	GPIO1[27]	—	
		TSEC2_RXD[5]	GPIO1[28]	—	
		TSEC2_RXD[4]	GPIO1[29]	—	
13	TSEC2_E	TSEC2_RX_ER	GPIO1[25]	—	0 TBI 1 RTBI, Else

Table 5-28. SICRH Bit Settings (continued)

SICRH[Bits] Value:		0b0/0b00	0b1/0b01	0b10	Reset Value
Bits	Group	Pin Function 0	Pin Function 1	Pin Function 2	
14	TSEC2_F	TSEC2_TXD[7]	GPIO1[31]	—	0 TBI 1 RTBI, Else
		TSEC2_TXD[6]	DR_TERM_SEL	—	
		TSEC2_TXD[5]	DR_OPMODE1	—	
		TSEC2_TXD[4]	DR_OPMODE0	—	
15	TSEC2_G	TSEC2_TX_ER	GPIO1[24]	—	0 TBI 1 RTBI, Else
16	TSEC2_H	TSEC2_TX_CLK	GPIO1[30]	—	0 TBI 1 RTBI, Else
17	GPIO2_A	$\overline{\text{IRQ}}$[0]/$\overline{\text{MCP_IN}}$	GPIO2[12]	—	0
18	GPIO2_B	$\overline{\text{IRQ}}$[1]	GPIO2[13]	—	0
19	GPIO2_C	$\overline{\text{IRQ}}$[2]	GPIO2[14]	—	0
20	GPIO2_D	$\overline{\text{IRQ}}$[3]	GPIO2[15]	—	0
21	GPIO2_E	$\overline{\text{IRQ}}$[4]	GPIO2[16]	—	0
22	GPIO2_F	$\overline{\text{IRQ}}$[5]	GPIO2[17]	—	0
23–24	GPIO2_G	$\overline{\text{IRQ}}$[6]	GPIO2[18]	$\overline{\text{CKSTOP_OUT}}$	0
25–26	GPIO2_H	$\overline{\text{IRQ}}$[7]	GPIO2[19]	$\overline{\text{CKSTOP_IN}}$	0
27[1]	PCI	PCI_CLK_OUT[3:4]	$\overline{\text{LCS}}$[6:7]	—	0
28–29	Reserved	Writes to SICRH[28–29] should preserve the reset value.			0
30	TSOBI1	See Table 5-29 for description and reset value.			
31	TSOBI2	See Table 5-29 for description and reset value.			

1 PCI_CLK[3:4]/$\overline{\text{LCS}}$[6:7] drive zeros during and after the period in which $\overline{\text{PORESET}}$ and $\overline{\text{HRESET}}$ are asserted. If a slave device is connected to PCI_CLK[3]/$\overline{\text{LCS}}$[6] or PCI_CLK[4]/$\overline{\text{LCS}}$[7] and this slave will drive data on LAD signals when it has $\overline{\text{LCS}}$ and $\overline{\text{LOE}}$ both asserted, this will lead to contention on the LAD bus and errors in loading the RCW data. It will also lead to errors when the e300 CPU will try to fetch boot data.
The user should take one of the following precautions:
Use an I^2C EEPROM to load RCW, and use the boot sequencer to set SICRH[27] before letting the CPU boot its code.
If possible, use a hard coded RCW and use the remote PCI host to set SICRH[27] before letting the CPU boot its code.
If possible, use LCS[6:7] multiplexed with SPI pins. See Section 5.3.2.5, “System I/O Configuration Register Low (SICRL).”
Do not connect local bus slaves to LCS[6:7].

Table 5-29. SICRH[30–31] Bit Settings

Bits	Name	Description	Reset Value
30	TSOBI1	TSEC1 output buffer impedance. This bit controls the output buffer impedance of the TSEC1 output signals, used for reduced pin mode interfaces (RTBI/RGMII). The output buffer impedance should be correlated to the voltage supplied to the TSEC1 I/O pins (LVDD1). 0 Output buffer is set for 40 ohm, 3.3 volts. 1 Output buffer is set for 40 ohm, 2.5 volts.	0 Else 1 RGMII, RTBI
31	TSOBI2	TSEC2 output buffer impedance. This bit controls the output buffer impedance of TSEC2 output signals, used for reduced pin mode interfaces (RTBI/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 ohm, 3.3 volts. 1 Output buffer is set for 40 ohm, 2.5 volts.	0 Else 1 RGMII, RTBI

5.3.2.7 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)”).

5.3.2.7.1 DDR Debug Configuration

The DDR debug configuration enables a DDR memory controller to either debug mode in which the DDR SDRAM source ID field and data valid strobe are driven onto one of two optional sets of pins:

1. ECC pins. ECC checking and generation are disabled in this case. ECC signals driven from the SDRAMs must be electrically disconnected from the ECC I/O pins of the processor in this mode. Set SICRH[0] to select this mode.
2. UART pins. UART operation is disabled, and any signals driven by UART devices must be electrically disconnected from the UART I/O pins. Set SICRL[4–5] to 0b01 to select this mode.

5.3.2.7.2 Local Bus Debug Configuration

The local bus debug configuration enables a LBC debug mode in which the SDRAM source ID field and data valid strobe for LBC memory accesses are driven onto UART pins. UART operation must be disabled, and any signals driven by UART devices must be electrically disconnected from the UART I/O pins in this case. Set SICRL[4–5] to 0b10 to select this mode.

5.3.2.8 DDR Control Driver Register (DDRCDR)

The DDR control driver register (DDRCDR) contains bits that allow control over the driver of the DDR SDRAM controller. Note that the MDIC signals require the use of precision 18-Ω resistors; MDIC0 should be pulled to GND, while MDIC1 should be pulled to GV_{DD}.

The fields in DDRCDR (other than ODT) are used to enable driver calibration with the MDIC[0:1] signals. This can be used to calibrate the DDR drivers to 18 ohms. However, this should only be used for full-strength driver applications. If half strength is desired, then this calibration should remain disabled.

Hardware DDR driver calibration is enabled using DDRCDR[DHC_EN]. If hardware calibration is used, then it should be set before DDR_SDRAM_CFG[MEM_EN] is set.

Software can be used to calibrate the drivers instead of the automatic hardware calibration. If software calibration is used, the following steps should be taken:

1. Set DDRCDR[DSO_EN] and ensure that DDRCDR[DHC_EN] is cleared.
2. Set the highest impedance (value 0000) for DDRCDR[DSO_PZ].
3. Set DDRCDR[MDIC0_OE] to enable the output enable for MDIC[0].
4. After at least 4 cycles, read DDRDSR[0]. If the value is 0, then use the next lower impedance, and read DDRDSR[0] again. Once a value of 1 is detected, then leave DDRCDR[DSO_PZ] at the calibrated value.
5. Clear DDRCDR[MDIC0_OE].
6. After DDRCDR[DSO_PZ] is calibrated, set a value of 0000 for DDRCDR[DSO_NZ]
7. Set DDRCDR[MDIC1_OE] to enable the output enable for MDIC[1]
8. After at least 4 cycles, read DDRDSR[1]. If the value is 1, then use the next lower impedance, and read DDRDSR[1] again. Once a value of 0 is detected, then leave DDRCDR[DSO_NZ] at the calibrated value
9. Clear DDRCDR[MDIC1_OE]

Note that the legal impedance values, from highest impedance to lowest impedance, are as follows:

- 0000
- 1000
- 1100
- 1110
- 1111

DDRCDR is shown in Figure 5-16.

Offset 0x00128 Access: Read/Write

	0	1	2	5	6	9	10	12	13	14	15
R W	DHC_EN	DSO_EN	DSO_PZ	DSO_NZ		MDIC0_OE	MDIC0_OE	ODT	DDR_TYPE	—	
Reset	0	1	1 1	0 0	1 1	0 0	0 0	0	1	0	0

	16	29	30	31
R W	—			DRQ
Reset	0 0 0 0	0 0 0 0	0 0 0 0	0	0	0	1

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

Table 5-30 shows the bit definition of the DDRCDR.

Table 5-30. DDRCDR Field Descriptions

Bits	Name	Description
0	DHC_EN	DDR driver hardware compensation enable
1	DSO_EN	DDR driver software override enable
2–5	DSO_PZ	DDR driver software p-impedance override 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	DSO_NZ	DDR driver software n-impedance override 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	MDIC0_OE	Output enable control for MDIC0 pin when software impedance calibration is performed. 0 - MDIC0 is disabled 1 - MDIC0 is enabled
11	MDIC1_OE	Output enable control for MDIC1 pin when software impedance calibration is performed. 0 - MDIC1 is disabled 1 - MDIC1 is enabled
12	ODT	ODT termination value for IOs 0 75 Ohm 1 150 Ohm
13	DDR_TYPE	Selects voltage level for DDR pads 0 DDR2 (1.8V mode) nominal impedance-18 Ohm 1 DDR1 (2.5V mode) nominal impedance-18 Ohm
14–30	—	Reserved
31	DRQ	Drain queue before sleep

5.3.2.9 DDR Debug Status Register (DDRDSR)

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

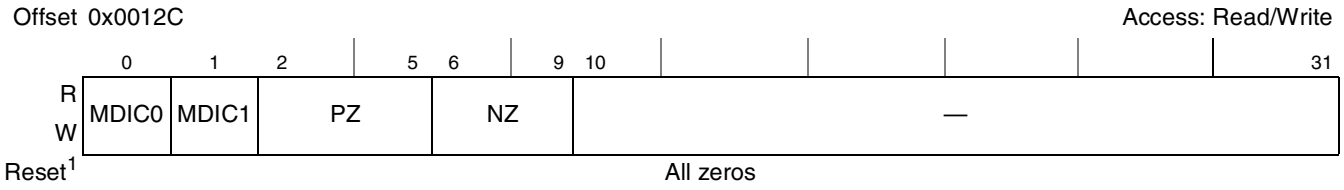


Figure 5-17. DDR Debug Status Register (DDRDSR)

[1] Reset value of bits 0-9 depends on the actual state of the signals being monitored.

Table 5-31 shows the bit settings of the DDRDSR.

Table 5-31. DDRDSR Field Descriptions

Bits	Name	Description
0	MDIC0	DDR driver compensation input value read back for MDIC0. Note that MDIC0 is the PFET driver impedance calibration pin.
1	MDIC1	DDR driver compensation input value read back for MDIC1. Note that MDIC1 is the NFET driver impedance calibration pin.
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

5.4 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 serves. 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.

5.4.1 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 5-18 shows a high-level block diagram of the WDT.

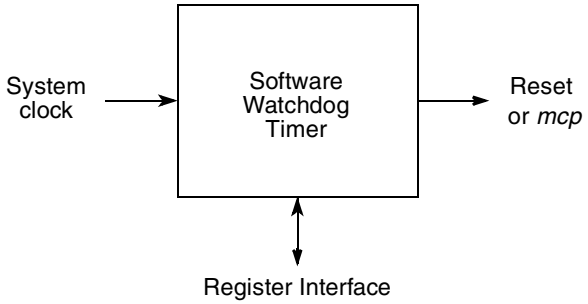


Figure 5-18. Software Watchdog Timer High-Level Block Diagram

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.

5.4.2 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 ~12.8-sec. maximum software time-out delay for 333-MHz input clock
- Functional and programming compatibility with MPC8260 watchdog timer

5.4.3 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.

5.4.4 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; 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 2, "Memory Map."

Table 5-32 shows the WDT memory map.

Table 5-32. WDT Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
0x0–0x3	Reserved	—	—	—
0x4	System watchdog control register (SWCRR)	R/W	0x0000_0007	5.4.4.1/5-33
0x8	System watchdog count register (SWCNR)	R	0x0000_FFFF	5.4.4.2/5-34
0xC–0xD	Reserved	—	—	—
0xE	System watchdog service register (SWSRR)	R/W	0x0000_0000	5.4.4.3/5-34

5.4.4.1 System Watchdog Control Register (SWCRR)

The system watchdog control register (SWCRR), shown in Figure 5-19, controls the software watchdog period and configures watchdog timer operation. SWCRR can be read at any time but can be written only once after system reset.

Offset 0x4 Access: Read/Write

	0				15	16							28	29	30	31
R W	SWTC					—								SWEN	SWRI	SWPR
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0[1]	1	1

[1] Special: SWCRR[SWEN] reset value directly depends on RCWHR[SWEN] (reset configuration word high).

Figure 5-19. System Watchdog Control Register (SWCRR)

Table 5-33 defines the bit fields of SWCRR.

Table 5-33. 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. **Note:** 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

Table 5-33. SWCRR Bit Settings (continued)

Bits	Name	Description
29	SWEN	Watchdog enable 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. 0 Watchdog timer disabled 1 Watchdog timer enabled **Note:** After software writes the SWRI bit, the state of SWEN cannot be changed.
30	SWRI	Software watchdog reset/interrupt select bit A WDT timer out 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.

5.4.4.2 System Watchdog Count Register (SWCNR)

The system watchdog count register (SWCNR), shown in Figure 5-20, provides visibility to the watchdog counter value. SWCNR is a read-only register. Writes to SWCNR have no effect and terminate without transfer error exception.

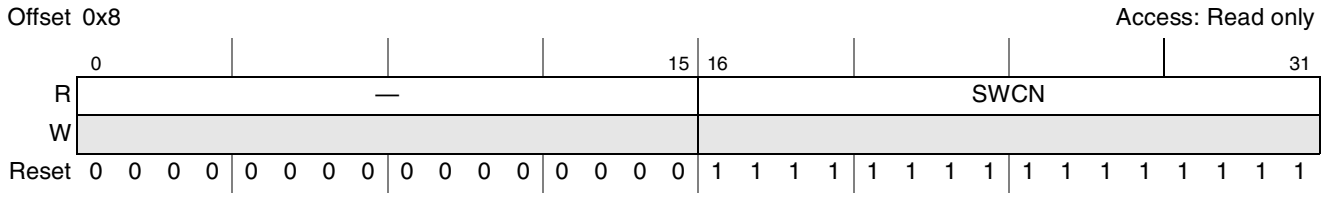


Figure 5-20. System Watchdog Count Register (SWCNR)

Table 5-34 defines the bit fields of SWCNR.

Table 5-34. 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. **Note:** Reading the 16 least-significant bits of 32-bit SWCNR register with two 8-bit reads is not guaranteed to return a coherent value.

5.4.4.3 System Watchdog Service Register (SWSRR)

The system watchdog service register (SWSRR) is shown in Figure 5-21. 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.

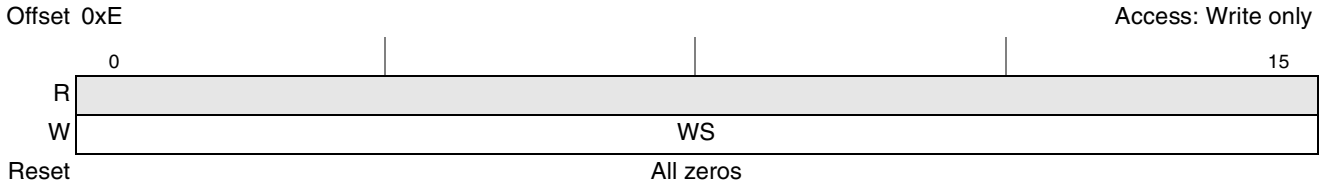


Figure 5-21. System Watchdog Service Register (SWSRR)

Table 5-35 defines the bit fields of SWCNR.

Table 5-35. 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.

5.4.5 Functional Description

5.4.5.1 Software Watchdog Timer Unit

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 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 5-22 shows a state diagram for the watchdog timer.

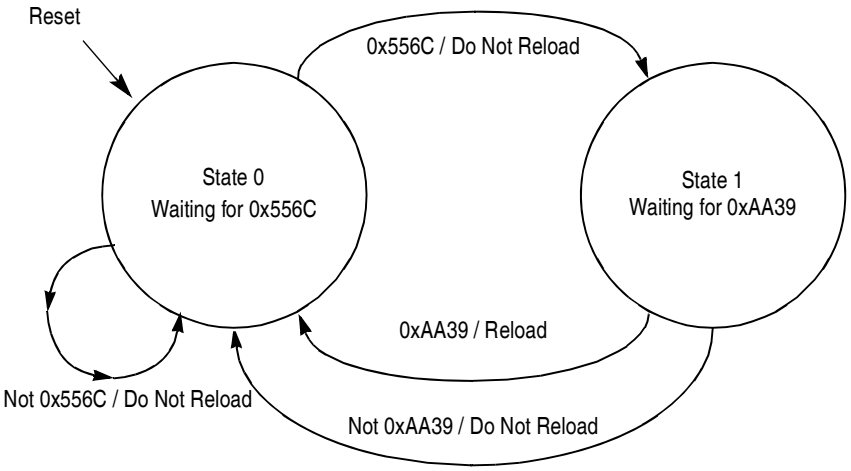


Figure 5-22. Software Watchdog Timer Service State Diagram

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 5-23 shows how to handle this need.

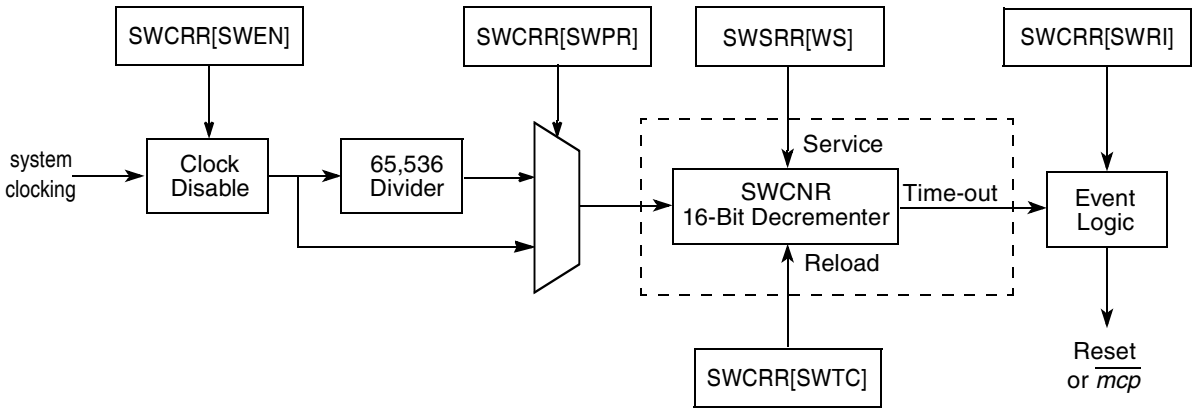


Figure 5-23. Software Watchdog Timer Functional Block Diagram

Figure 5-23 shows that the range is determined by SWCRR[SWTC]. The value in SWTC is then loaded into a 16-bit decrementer clocked by the system clock. An additional divide-by-65,536 prescaler value is used when needed.

The decrementer 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.

5.4.5.2 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)

    This is the default value after soft reset.

  — 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 soft reset or machine check interrupt to the core.

  — Reset mode (SWCRR[SWRI] = 1).

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

5.4.6 Initialization/Application Information

5.4.6.1 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 (~12.8 sec. for a 333-MHz system clock).

- WDT initial servicing

  If the software watchdog timer is to be used, the special service sequence, described in Section 5.4.5.1, “Software Watchdog Timer Unit,” must be executed after system reset and not later than the first WDT time-out (~12.8 sec. for 333 MHz system clock).

  Subsequently, periodical WDT servicing should be performed according to the programming guidelines given in Section 5.4.5.1, “Software Watchdog Timer Unit.”

5.5 Real Time Clock Module (RTC)

The following sections describe the theory of operation of the real time clock module (RTC) 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 Overview

The device platform provides a real time clock (RTC) timer suitable for timestamping or time and calendar generation. It can maintain a one-second 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 5-24 shows the high level RTC block diagram.

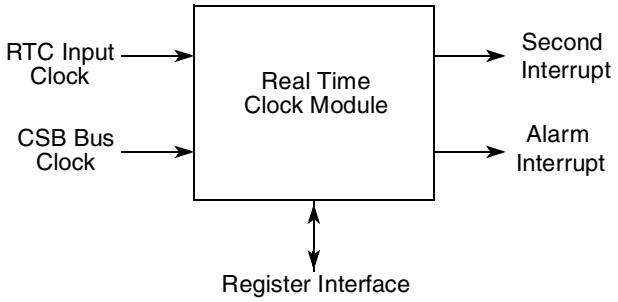


Figure 5-24. Real Time Clock Module High Level Block Diagram

5.5.2 Features

The key features of the RTC include the following:

- Maintains a one-second count, unique over a period of thousand of years.
- 32-bit RTC counter can be initialized by software to specific initial count value.
- Provides an alarm function with programmable and maskable alarm interrupt.
- Provides programmable and maskable every second interrupt.
- Uses two possible clock sources: the CSB bus clock or an external RTC clock.
- RTC function can be disabled if needed.

5.5.3 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.

5.5.4 External Signal Description

The following sections provide an overview and detailed descriptions of the RTC signals.

5.5.4.1 Overview

There is one distinct external RTC clock input signal, defined in Table 5-36.

Table 5-36. RTC Signal Properties

Name	Port	Function	I/O	Reset	Pull Up
RTC_CLK	RTC_CLK	Real time clock input.	I	N/A	—

5.5.4.2 Detailed Signal Descriptions

Table 5-37 provides a detailed description of the external RTC signal.

Table 5-37. RTC External Signal—Detailed Signal Description

Signal	I/O	Description	
RTC_CLK	I	This signal is used as the timebase for the real time clock module.	
		State Meaning	—
		Timing	—

5.5.5 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 a 32-bit quantities.

All addresses used in this section are offsets from the RTC base, as defined in Chapter 2, "Memory Map."

Table 5-32 shows the memory map of the RTC.

Table 5-38. RTC Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
0x00	Real time counter control register (RTCNR)	R/W	0x0000_0000	5.5.5.1/5-40
0x04	Real time counter load register (RTLDR)	R/W	0x0000_0000	5.5.5.2/5-41
0x08	Real time counter prescale register (RTPSR)	R/W	0x0000_0000	5.5.5.3/5-41
0x0C	Real time counter register (RTCTR)	R	0x0000_0000	5.5.5.4/5-42
0x10	Real time counter event register (RTEVR)	w1c	0x0000_0000	5.5.5.5/5-42
0x14	Real time counter alarm register (RTALR)	R/W	0xFFFF_FFFF	5.5.5.6/5-43
0x18–0x1F	Reserved	—	—	

5.5.5.1 Real Time Counter Control Register (RTCNR)

The real time counter control register (RTCNR), shown in Figure 5-25, is used to enable RTC functions. The register can be read at any time.

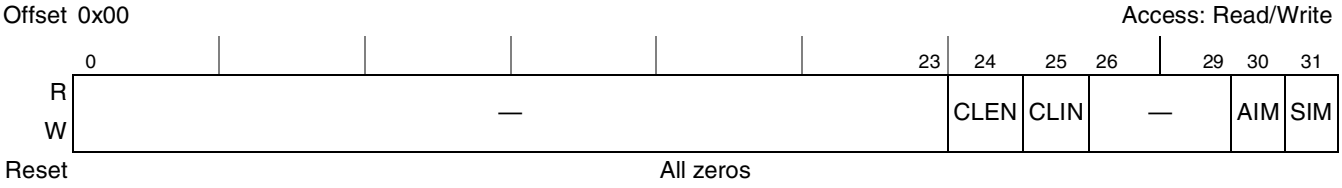


Figure 5-25. Real Time Counter Control Register (RTCNR)

Table 5-39 defines 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.

Table 5-39. RTCNR Bit Settings (continued)

Bits	Name	Description
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 periodic interrupt timer is CSB input clock. 1 The input clock to the periodic interrupt timer is the external RTC clock.
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.

5.5.5.2 Real Time Counter Load Register (RTLDR)

The real time counter load register (RTLDR), shown in Figure 5-26, contains the 32-bit value to be loaded in the 32-bit RTC counter.

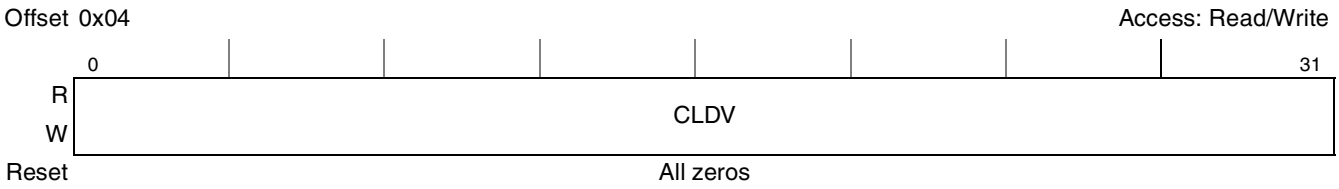


Figure 5-26. Real Time Counter Load Register (RTLDR)

Table 5-40 defines the bit fields 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.

5.5.5.3 Real Time Counter Prescale Register (RTPSR)

The real time counter prescale register (RTPSR), shown in Figure 5-27, is a read/write register used to configure the RTC prescaler's value.

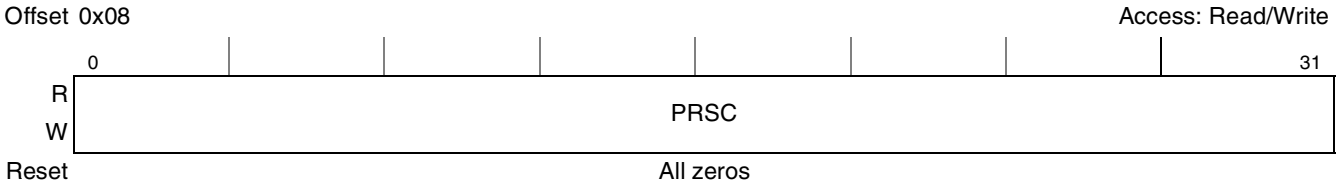


Figure 5-27. Real Time Counter Prescale Register (RTPSR)

Table 5-41 defines the bit fields of RTPSR.

Table 5-41. RTPSR Bit Settings

Bits	Name	Description
0–31	PRSC	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. 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. System reset and the loading of a new value into the counter both reset the prescaler counter. Clearing RTCNR[CLE] stops the prescaler counter.

5.5.5.4 Real Time Counter Register (RTCTR)

The real time counter register (RTCTR), shown in Figure 5-28, 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.

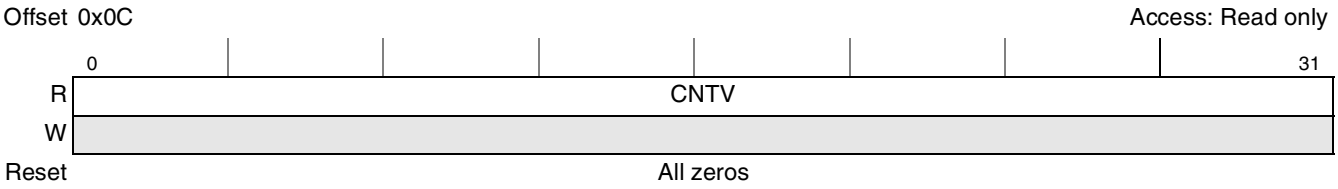


Figure 5-28. Real Time Counter Register (RTCTR)

Table 5-42 defines the bit fields 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].

5.5.5.5 Real Time Counter Event Register (RTEVR)

The real time counter event register (RTEVR), shown in Figure 5-29, is used to report the source of the interrupts. The register can be read at any time.

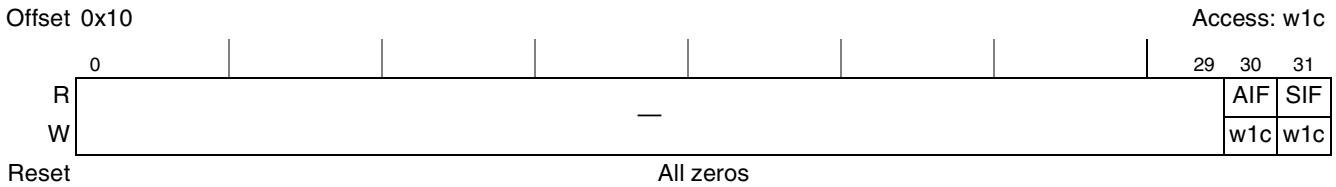


Figure 5-29. Real Time Counter Event Register (RTEVR)

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

Table 5-43 defines 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 issues an interrupt after the RTC counter counts to zero.
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 and should be cleared by software.

5.5.5.6 Real Time Counter Alarm Register (RTALR)

The real time counter alarm register (RTALR), shown in Figure 5-30, contains the 32-bit alarm (ALRM) value. When the value of the RTC counter equals the RTALR[ALRM] value, a maskable interrupt is generated.

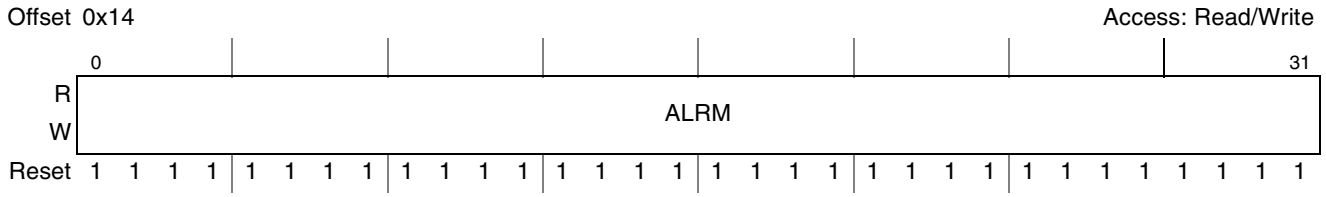


Figure 5-30. Real Time Counter Alarm Register (RTALR)

Table 5-44 defines the bit fields 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].

5.5.6 Functional Description

5.5.6.1 Real Time Counter Unit

The real time clock (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 if 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 an 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 the 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 hard reset but is not affected by soft reset. It is initialized by the software. The RTC function can be disabled.

Figure 5-31 shows the functional RTC block diagram.

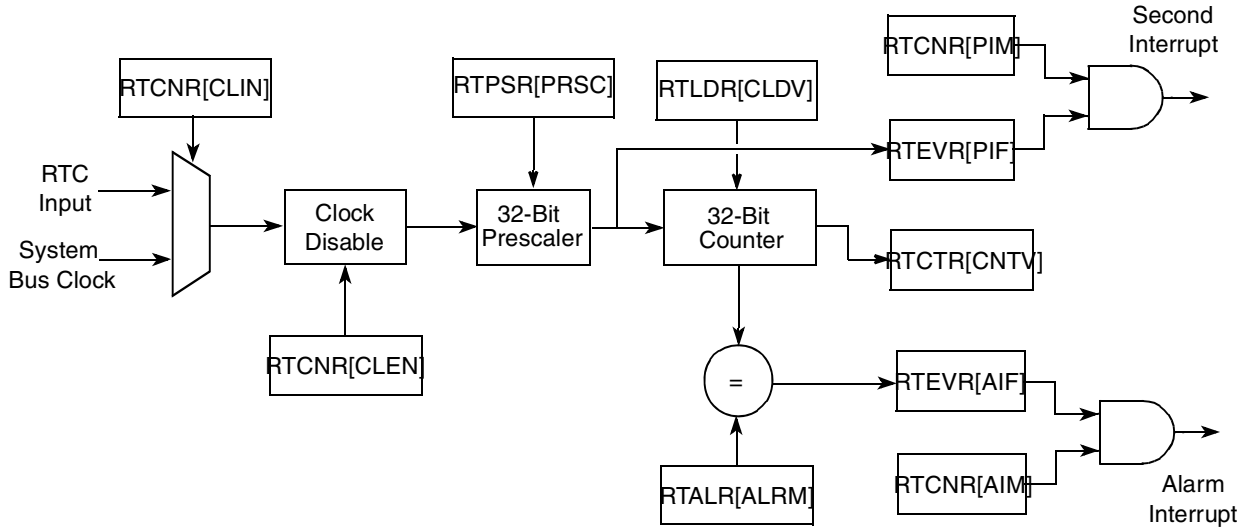


Figure 5-31. Real Time Clock Module Functional Block Diagram

5.5.6.2 RTC Operational Modes

The RTC unit can operate in the following modes:

- RTC enable/disable mode:

  RTCNR[CLEN] enables the RTC timer. It should be set by software after a system reset to enable the RTC timer.

  — RTC disable mode (RTCNR[CLEN] = 0)

  When the RTC's clock is disabled, counter maintains its old value (default).

  — RTC enable mode (RTCNR[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 (RTCNR[SIM] = 1)

  In this mode the RTC set the RTEVR[SIF] flag and generate an interrupt after the RTC's 32-bit counter reaches zero.

  — RTC every-second interrupt disable mode (RTCNR[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 (RTCNR[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 (RTCNR[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 32.768 kHz crystal.

— RTC uses the internal input clock mode (RTCNR[CLIN] = 0)

— RTC uses the external 32.768 kHz crystal clock (RTCNR[CLIN] = 1)

5.5.7 RTC Programming Guidelines

The following initialization sequence for the RTC is recommended:

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, RTC clock enable.

5.6 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.

5.6.1 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 a 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 5-32 shows the functional PIT block diagram.

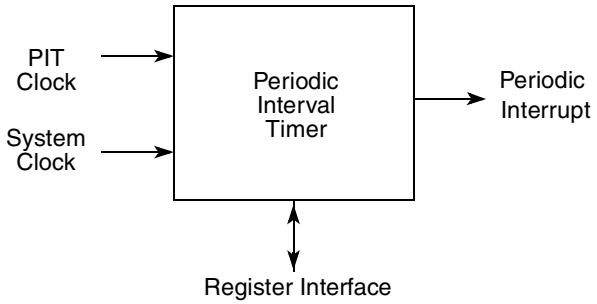


Figure 5-32. Periodic Interval Timer High Level Block Diagram

5.6.2 Features

The key features of the PIT include the following:

- Maintains a 32-bit down-counter, clocked by a 16-bit prescaled input clock
- 32-bit PIT counter can be initialized by software to specific initial count value.
- Provides programmable and maskable periodic interrupt
- Provides maximum period of ~9.5 days (for 333 MHz system clock)
- Uses two possible clock sources: the CSB clock or an external PIT clock
- PIT function can be disabled

5.6.3 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

5.6.4 External Signal Description

The following subsections provide an overview and detailed descriptions of the PIT signals.

5.6.4.1 Overview

There is one distinct external input signal (PIT clock), defined in Table 5-36.

Table 5-45. PIT Signal Properties

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

5.6.4.2 Detailed Signal Description

Table 5-37 describes of the external PIT signal.

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

Signal	I/O	Description	
PIT_CLK	I	This signal is used as the timebase for the periodic interval timer module.	
		State Meaning	—
		Timing	—

5.6.5 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 2, "Memory Map."

Table 5-32 shows the PIT memory map.

Table 5-47. PIT Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
0x00	Periodic interval timer control register (PTCNR)	R/W	0x0000_0000	5.6.5.1/5-47
0x04	Periodic interval timer load register (PTLDR)	R/W	0x0000_0000	5.6.5.2/5-48
0x08	Periodic interval timer prescale register (PTPSR)	R/W	0x0000_0000	5.6.5.3/5-49
0x0C	Periodic interval timer counter register (PTCTR)	R	0x0000_0000	5.6.5.4/5-49
0x10	Periodic interval timer event register (PTEVR)	w1c	0x0000_0000	5.6.5.5/5-50
0x14–0x1F	Reserved	—	—	

5.6.5.1 Periodic Interval Timer Control Register (PTCNR)

The periodic interval timer control register (PTCNR), shown in Figure 5-33, is used to enable the different PIT functions. The register can be read at any time.

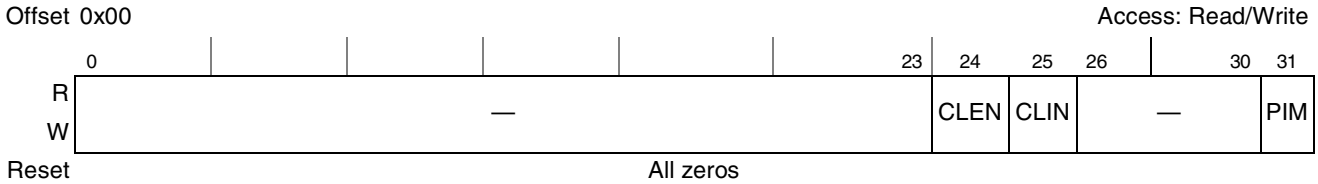


Figure 5-33. Periodic Interval Timer Control Register (PTCNR)

Table 5-39 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.
25	CLIN	Input clock control bit. The input clock to the PIT can be either an internal system clock or an external 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 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.

5.6.5.2 Periodic Interval Timer Load Register (PTLDR)

The periodic interval timer load register (PTLDR), shown in Figure 5-34, contains the 32-bit value to be loaded in a 32-bit PIT counter.

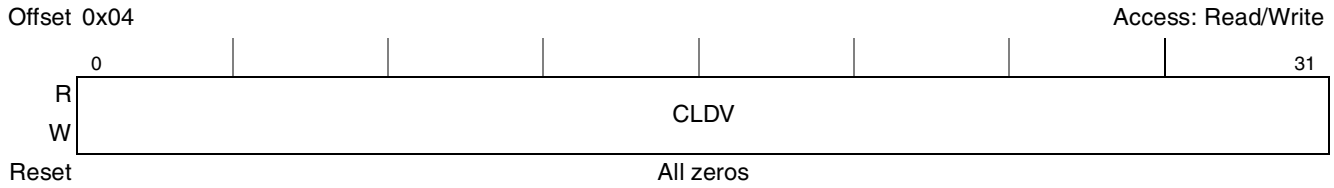


Figure 5-34. Periodic Interval Timer Load Register (PTLDR)

Table 5-40 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.

5.6.5.3 Periodic Interval Timer Prescale Register (PTPSR)

The periodic interval timer prescale register (PTPSR), shown in Figure 5-35, is a read/write register that used to configure the PIT prescaler's value.

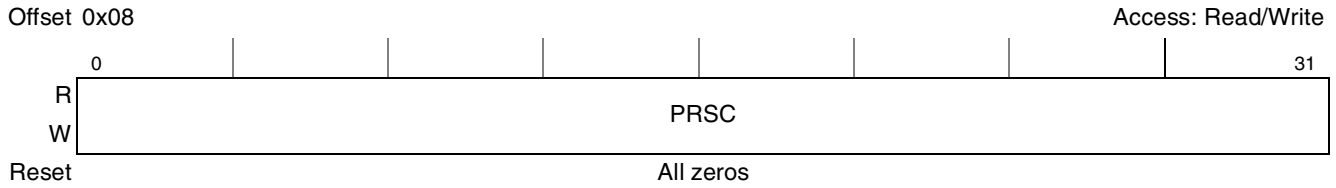


Figure 5-35. Periodic Interval Timer Prescale Register (PTPSR)

Table 5-41 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.

5.6.5.4 Periodic Interval Timer Counter Register (PTCTR)

The periodic interval timer counter register (PTCTR), shown in Figure 5-36, is a read-only register that shows the current value in the PIT counter. The PTCTR counter is not affected by reads or writes.

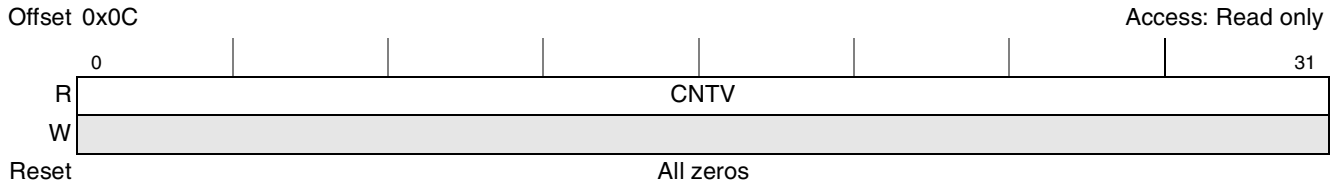


Figure 5-36. Periodic Interval Timer Counter Register (PTCTR)

Table 5-42 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].

5.6.5.5 Periodic Interval Timer Event Register (PTEVR)

The periodic interval timer event register (PTEVR), shown in Figure 5-37, 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.

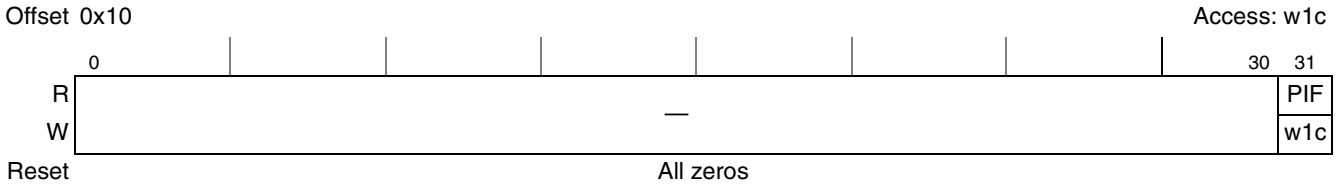


Figure 5-37. Periodic Interval Timer Event Register (PTEVR)

Table 5-43 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. Its asserted if the PIT issues an interrupt after the SPMPIT counter counts to zero. This status bit should be cleared by software.

5.6.6 Functional Description

5.6.6.1 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 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 5-38 shows the functional PIT block diagram.

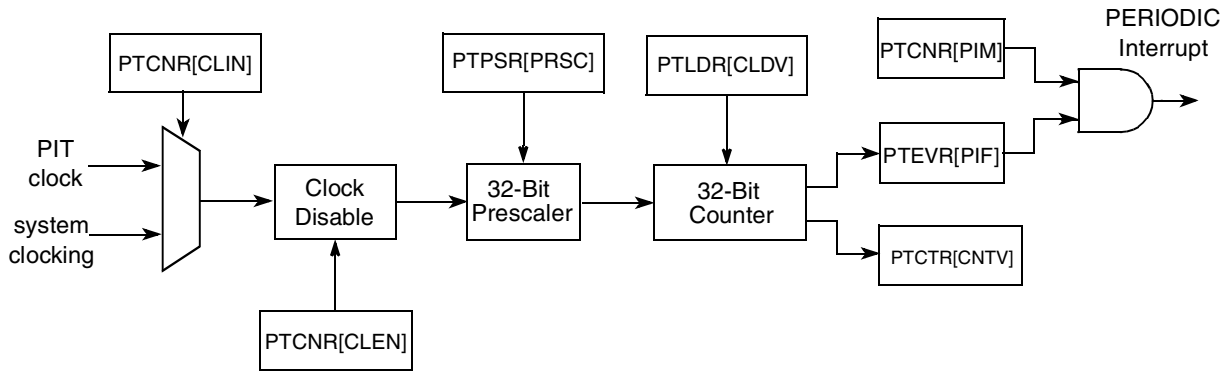


Figure 5-38. Periodic Interval Timer Functional Block Diagram

5.6.6.2 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 PIT clock.

  — PIT use the internal input clock mode (PTCNR[CLIN] = 0)

  — PIT use the PIT clock (PTCNR[CLIN] = 1)

5.6.7 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.

See Section 5.5.7, "RTC Programming Guidelines," for real-time clock programming guidelines.

5.7 General-Purpose Timers (GTMs)

The following sections describe theory of operation of the general purpose (global) timers, including a definition of the external signals and the functions they serve. 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.

5.7.1 Overview

The global timer module (GTM) 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 5-24 shows the functional GTM block diagram.

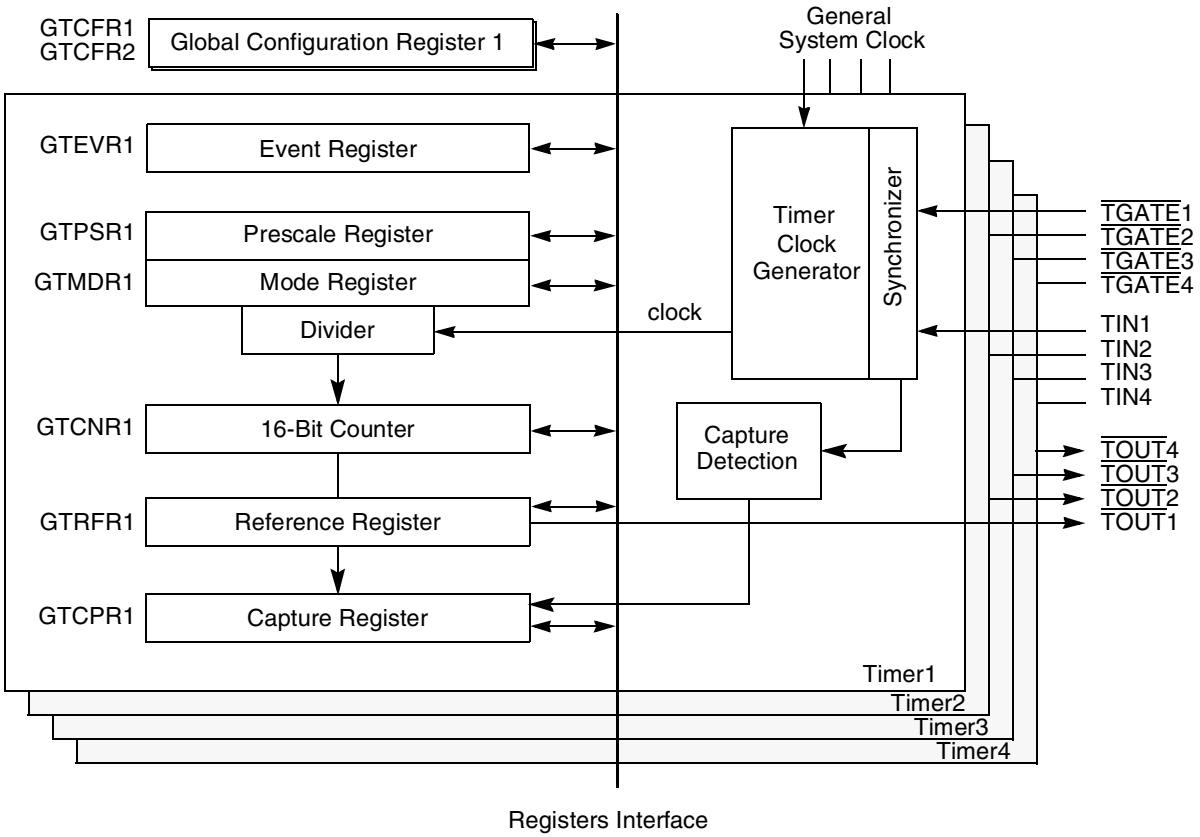


Figure 5-39. Global Timers Block Diagram

5.7.2 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 to form a 32-bit timer
- One timer cascaded internally or externally to form a 64-bit timer
- Maximum period of ~50 msecond (at 333 MHz bus clock) for 16-bit timer
- Maximum period of ~12.8 second (at 333 MHz bus clock) for 32-bit timer
- Maximum period of thousand of year (at 333 MHz bus clock) for 64-bit timer
- 3-nanosecond timer resolution (at 333 MHz bus clock)
- 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

5.7.3 Modes of Operation

The GTM unit can operate in the following modes:

5.7.3.1 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.

5.7.3.2 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 TINx pin

5.7.3.3 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.

5.7.3.4 Capture Modes

Each timer has a 16-bit field in GTCPR, used to latch the value of the counter when a defined transition of TINx 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.

5.7.4 External Signal Description

The following sections provide an overview and detailed descriptions of the GTM signals.

5.7.4.1 Overview

There are four distinct external input timer capture signals (TIN1, TIN2, TIN3 and TIN4), four distinct external input timer get signals ($\overline{\text{TGATE}}$1, $\overline{\text{TGATE}}$2, $\overline{\text{TGATE}}$3 and $\overline{\text{TGATE}}$4), and four distinct external timer output signals ($\overline{\text{TOUT}}$1, $\overline{\text{TOUT}}$2, $\overline{\text{TOUT}}$3 and $\overline{\text{TOUT}}$4). 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

Table 5-53. GTM Signal Properties (continued)

Name	Port	Function	I/O	Reset	Require Pull Up
TIN4	TIN4	Global timer 4 capture control signal	I	0	No
$\overline{\text{TGATE}}$1	$\overline{\text{TGATE}}$1	Global timer 1 counter gate control signal	I	0	No
$\overline{\text{TGATE}}$2	$\overline{\text{TGATE}}$2	Global timer 2 counter gate control signal	I	0	No
$\overline{\text{TGATE}}$3	$\overline{\text{TGATE}}$3	Global timer 3 counter gate control signal	I	0	No
$\overline{\text{TGATE}}$4	$\overline{\text{TGATE}}$4	Global timer 4 counter gate control signal	I	0	No
$\overline{\text{TOUT}}$1	$\overline{\text{TOUT}}$1	Global timer 1 counter output signal	O	1	No
$\overline{\text{TOUT}}$2	$\overline{\text{TOUT}}$2	Global timer 2 counter output signal	O	1	No
$\overline{\text{TOUT}}$3	$\overline{\text{TOUT}}$3	Global timer 3 counter output signal	O	1	No
$\overline{\text{TOUT}}$4	$\overline{\text{TOUT}}$4	Global timer 4 counter output signal	O	1	No

5.7.4.2 Detailed Signal Descriptions

Table 5-54 provides detailed descriptions of the external GTM signals.

Table 5-54. GTM External Signals—Detailed Signal Descriptions

Signal	I/O	Description	
TIN*n*	I	Global timer capture control signal. Used to latch the value of the counter when a defined transition of TIN*n* is sensed by the corresponding input capture edge detector.	
		State Meaning	Asserted/Negated —According to the programmed polarity by the corresponding GTMDR*n*[CE]). Each timer has a 16-bit 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. Upon a capture or reference event, the corresponding GTEVR bit is set and a maskable interrupt request is issued to the interrupt controller.
		Timing	Assertion/Negation—Asynchronous to internal bus clock. TIN*n* is internally synchronized to the system bus clock. If 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.	
		State Meaning	Asserted/Negated—According to the programmed polarity by the corresponding GTCFR[GMx] bits. In a reset gate mode (GTCFR[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 (GTCFR[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 GTCNR*n*[CNV*n*].
		Timing	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 after one system bus clock when working with the internal clock.

Table 5-54. GTM External Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
$\overline{\text{TOUT}}n$	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.	
		State Meaning	Asserted/Negated—According to the programmed polarity by the corresponding GTMDR*n*[OM*n*]. 1. Active-low pulse on $\overline{\text{TOUT}}n$ for one timer input clock cycle as defined by the GTMDR*n*[ICLK*n*] bits (GTMDR*n*[OM*n*] = 1). Thus, $\overline{\text{TOUT}}n$ may be low for one general system clock period, one general system slow go clock period, or one TIN*n* pin clock cycle period. 2. Toggle the $\overline{\text{TOUT}}n$ pin (GTMDR*n*[OM*n*] = 0). $\overline{\text{TOUT}}n$ changes occur on the rising edge of the system clock.
		Timing	Assertion/Negation—$\overline{\text{TOUT}}n$ changes occur on the rising edge of the system clock.

5.7.5 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 GPT Base, as defined in Chapter 2, “Memory Map.”

Table 5-55 shows memory map of the GTM.

Table 5-55. GTM Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
0x00	Timer 1 and 2 global timers configuration register (GTCFR1)	R/W	0x00	5.7.5.1/5-57
0x01–0x03	Reserved	—	—	—
0x04	Timer 3 and 4 global timers configuration register (GTCFR2)	R/W	0x00	5.7.5.1/5-57
0x05–0x0F	Reserved	—	—	—
0x10	Timer 1 global timers mode register (GTMDR1)	R/W	0x0000	5.7.5.2/5-60
0x12	Timer 2 global timers mode register (GTMDR2)			
0x14	Timer 1 global timers reference register (GTRFR1)	R/W	0xFFFF	5.7.5.3/5-62
0x16	Timer 2 global timers reference register (GTRFR2)			
0x18	Timer 1 global timers capture register (GTCPR1)	R/W	0x0000	5.7.5.4/5-62
0x1A	Timer 2 global timers capture register (GTCPR2)			
0x1C	Timer 1 global timers counter register (GTCNR1)	R/W	0x0000	5.7.5.5/5-63
0x1E	Timer 2 global timers counter register (GTCNR2)			
0x20	Timer 3 global timers mode register (GTMDR3)	R/W	0x0000	5.7.5.2/5-60
0x22	Timer 4 global timers mode register (GTMDR4)			

Table 5-55. GTM Register Address Map (continued)

Offset	Register	Access	Reset Value	Section/ Page
0x24	Timer 3 global timers reference register (GTRFR3)	R/W	0xFFFF	5.7.5.3/5-62
0x26	Timer 4 global timers reference register (GTRFR4)			
0x28	Timer 3 global timers capture register (GTCPR3)	R	0x0000	5.7.5.4/5-62
0x2A	Timer 4 global timers capture register (GTCPR4)			
0x2C	Timer 3 global timers counter register (GTCNR3)	R/W	0x0000	5.7.5.5/5-63
0x2E	Timer 4 global timers counter register (GTCNR4)			
0x30	Timer 1 global timers event register (GTEVR1)	w1c	0x0000	5.7.5.6/5-63
0x32	Timer 2 global timers event register (GTEVR2)			
0x34	Timer 3 global timers event register (GTEVR3)			
0x36	Timer 4 global timers event register (GTEVR4)			
0x38	Timer 1 global timers prescale register (GTPSR1)	R/W	0x0003	5.7.5.7/5-64
0x3A	Timer 2 global timers prescale register (GTPSR2)			
0x3C	Timer 3 global timers prescale register (GTPSR3)			
0x3E	Timer 4 global timers prescale register (GTPSR4)			

5.7.5.1 Global Timers Configuration Registers (GTCFR*n*)

The global timers configuration registers (GTCFR1 and GTCFR2), shown in Figure 5-40 and Figure 5-41, 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.

Offset 0x00 Access: Read/Write

	0	1	2	3	4	5	6	7
R W	PCAS	BCM	STP2	RST2	GM2	GM1	STP1	RST1
Reset	All zeros							

Figure 5-40. 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. **Note:** This bit is ignored in super-cascade mode (GTCFR2[SCAS]=1). **Note:** 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 will control the gate mode for timers 1 and 2 and GTCFR2[GM4] bit will control the gate mode for timers 3 and 4. GTCFR1[GM1] and GTCFR2[GM3] bits are ignored. 1 Normal operational mode
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]. **Note:** In backward compatible mode (GTCFR1[BCM]=0) this bit is ignored. GTCFR1[GM2] bit will control 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 5-41.

Offset 0x04 Access: Read/Write

	0	1	2	3	4	5	6	7
R W	PCAS	SCAS	STP4	RST4	GM4	GM3	STP3	RST3
Reset	All zeros							

Figure 5-41. 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	Pair-cascade mode 0 Normal operation. 1 Timers 3 and 4 cascade to form a 32-bit timer. **Note:** This bit is ignored in super-cascade mode (GTCFR2[SCAS]=1). **Note:** 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.
1	SCAS	Super cascade mode 0 Normal operation 1 Timers 1, 2, 3 and 4 cascade to form a 64-bit timer. **Note:** In super-cascade mode (GTCFR2[SCAS]=1) the pair-cascade mode bits are ignored, (GTCFR1/2[PCAS]=Don't Care). **Note:** 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.
2	STP4	Stop timer 4 0 Normal operation 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.
3	RST4	Reset timer 4 0 Reset the timer 4, including GTMDR4, GTRFR4, GTCNR4, GTCPR4 and GTEVR4 (a software reset is identical to an external reset). 1 Enable the corresponding timer if the STP4 bit is cleared.
4	GM4	Gate mode for $\overline{\text{TGATE4}}$ 0 Restart gate mode. The $\overline{\text{TGATE4}}$ is used to enable/disable count. A low level of $\overline{\text{TGATE4}}$ enables and a falling edge of $\overline{\text{TGATE4}}$ restarts the count (reset the dynamic counter's count value to 0) and a high level of $\overline{\text{TGATE4}}$ disables the count. 1 Normal gate mode. This mode is the same as 0, except the falling edge of $\overline{\text{TGATE4}}$ does not restart the appropriate count value in GTCNR4[CNV4].

Table 5-57. GTCFR2 Bit Settings (continued)

Bits	Name	Description
5	GM3	Gate mode for $\overline{\text{TGATE3}}$ 0 Restart gate mode. The $\overline{\text{TGATE3}}$ is used to enable/disable count. A low level of $\overline{\text{TGATE3}}$ enables and a falling edge of $\overline{\text{TGATE3}}$ restarts the count (reset the dynamic counter's count value to 0) and a high level of $\overline{\text{TGATE3}}$ disables the count. 1 Normal gate mode. This mode is the same as 0, except the falling edge of $\overline{\text{TGATE3}}$ does not restart the appropriate count value in GTCNR3[CNV3]. **Note:** In backward compatible mode (GTCFR1[BCM]=0) this bit is ignored. The GTCFR2[GM4] bit controls the gate mode for timers 3and 4.
6	STP3	Stop timer 3 0 Normal operation 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.
7	RST3	Reset timer 3 0 Reset the timer 3, including GTMDR3, GTRFR3, GTCNR3, GTCPR3 and GTEVR3 (a software reset is identical to an external reset). 1 Enable the corresponding timer if the STP3 bit is cleared.

5.7.5.2 Global Timers Mode Registers (GTMDR1–GTMDR4)

The global timers mode registers (GTMDR1, GTMDR2, GTMDR3 and GTMDR4) are shown in Figure 5-42.

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.

Offset 0x10(GTMDR1) 0x12(GTMDR2) 0x20(GTMDR3) 0x22(GTMDR4) Access: Read/Write

	0	7	8	9	10	11	12	13	14	15
R W	SPS		CE		OM	ORI	FRR	ICLK		GE
Reset	All zeros									

Figure 5-42. 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. **Note:** 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. **Note:** $\overline{\text{TOUT}}$*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 upon 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.
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.

5.7.5.3 Global Timers Reference Registers (GTRFR1–GTRFR4)

Global timers reference registers, shown in Figure 5-43, 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 GTCNR*n*[CNV] increments to the value in GTRFR*n*[TRV].

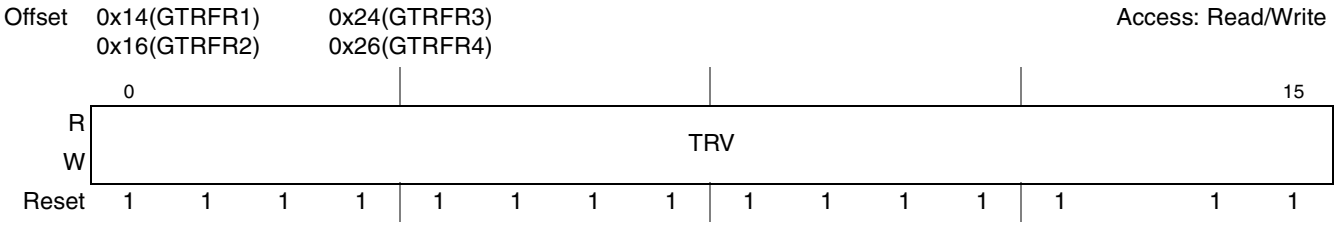


Figure 5-43. Global Timers Reference Registers (GTRFR1–GTRFR4)

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.

5.7.5.4 Global Timers Capture Registers (GTCPR1–GTCPR4)

Global timers capture registers (GTCPR1, GTCPR2, GTCPR3 and GTCPR4), shown in Figure 5-44, are used to latch the value of the counters according to GTMDR*n*[CE].

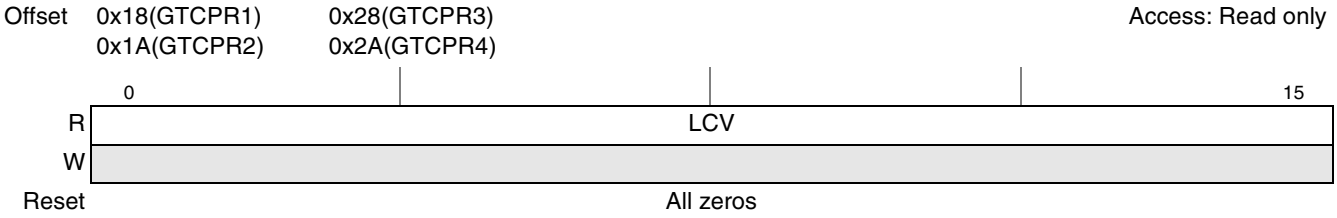


Figure 5-44. Global Timers Capture Registers (GTCPR1–GTCPR4)

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.

5.7.5.5 Global Timers Counter Registers (GTCNR1–GTCNR4)

Global timers counter registers (GTCNR1, GTCNR2, GTCNR3 and GTCNR4), shown in Figure 5-45, 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.

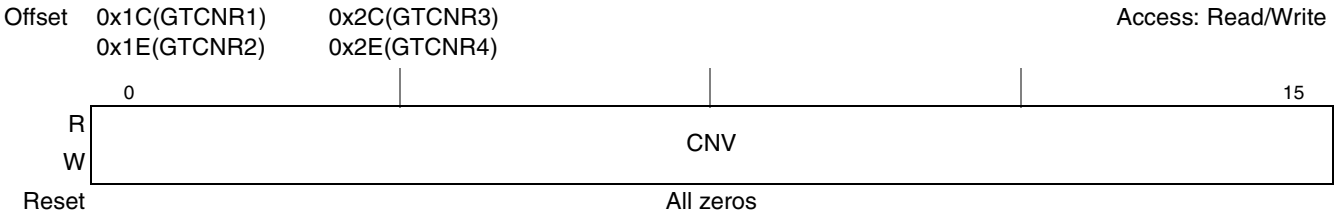


Figure 5-45. Global Timers Counter Registers (GTCNR1—GTCNR4)

Table 5-60 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.

5.7.5.6 Global Timers Event Registers (GTEVR1–GTEVR4)

Global timers event registers (GTEVR1, GTEVR2, GTEVR3 and GTEVR4), shown in Figure 5-46, 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.

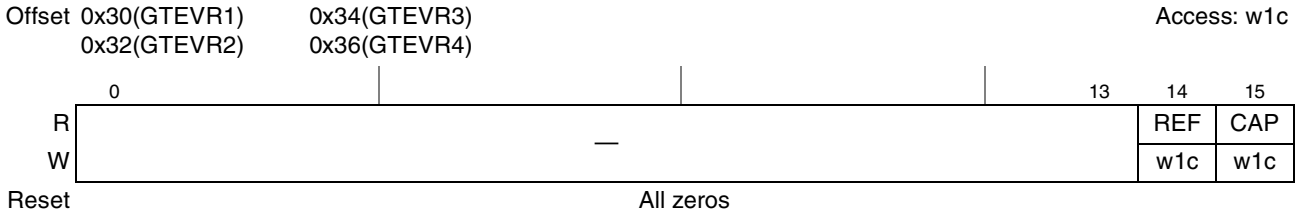


Figure 5-46. Global Timers Event Registers (GTEVR1—GTEVR4)

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.

5.7.5.7 Global Timers Prescale Registers (GTPSR1–GTPSR4)

The global timers prescale registers (GTPSR1, GTPSR2, GTPSR3 and GTPSR4) are shown in Figure 5-47.

Erratic behavior may occur if GTPSR*n* is not initialized before the corresponding GTMDR*n*.

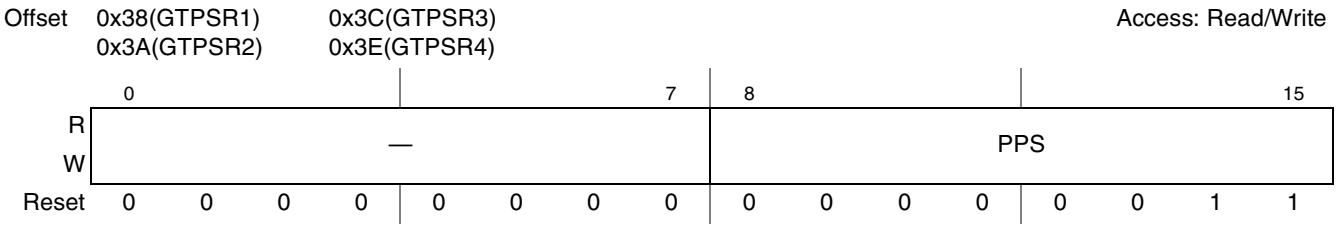


Figure 5-47. Global Timers Prescale Registers (GTPSR1–GTPSR4)

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:

$$GTMn_{prescaler} = (GTPSRn[PPS] + 1) \cdot (GTMDRn[SPS] + 1)$$

This gives a total prescale range from 1 (GTPSR*n*[PPS] = 0x00, GTMDR*n*[SPS] = 0x00) to 65,536 (GTPSR*n*[PPS] = 0xFF, GTMDR[SPS] = 0xFF).

5.7.6 Functional Description

5.7.6.1 General-Purpose Timer Units

The clock input to the timer's prescaler can be selected from the following sources:

- The system clock (*ipg_clock*)
- The system slow go clock (*ipg_clock* internally divided by 16)
- The corresponding TIN*n* signal, usually programmed in the general purpose I/O (GPIO) registers

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 65,537 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 (3 ns at a 333-MHz system clock, for example). The maximum period (when the reference value is all ones) for one 16-bit timer is ~50 ms at 333-MHz.

5.7.6.2 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 GTMRR 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 $\overline{\text{TOUT}}$*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.

5.7.6.3 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 ($\overline{\text{TGATE}}$*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 $\overline{\text{TGATE}}$ 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 $\overline{\text{TGATE}}$.

  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 $\overline{\text{TGATE}}$. The rising edge of $\overline{\text{TGATE}}$ completes the measurement and if $\overline{\text{TGATE}}$*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 $\overline{\text{TGATE}}$. 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 GTGCR.

NOTE

$\overline{\text{TGATE}}$ is internally synchronized to the system clock. If $\overline{\text{TGATE}}$ meets the asynchronous input setup time, the counter begins counting after one system clock when working with the internal clock.

5.7.6.4 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 GTGCF2[SCAS] = 0)

  If GTCFR*n*[PCAS] = 0 and GTCFR2[SCAS] = 0, the 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 5-48). When working in the none-cascaded mode, the non-cascaded GTRFR, GTCPR, and GTCNR should be referenced with appropriate 16-bit bus cycles.

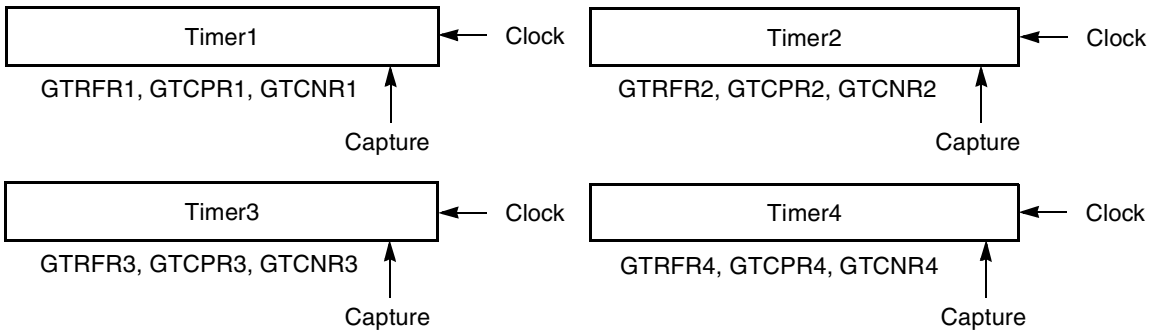


Figure 5-48. Timers Non-Cascaded Mode Block Diagram

- 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 5-49. 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 timer4) 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.

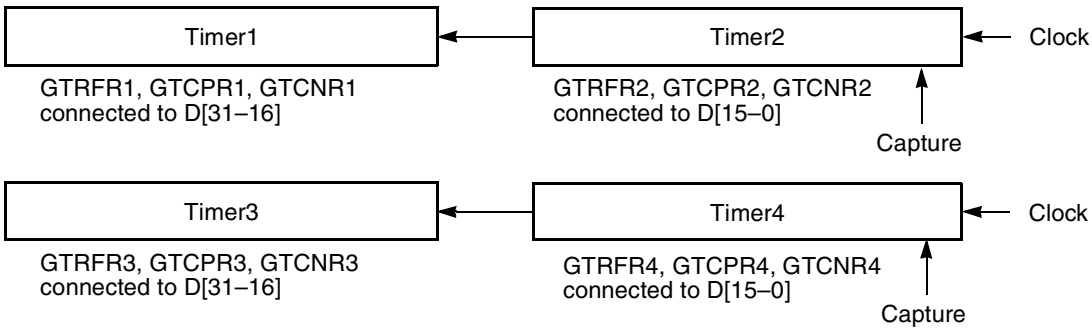


Figure 5-49. Timer Pair-Cascaded Mode Block Diagram

- 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 5-50.

  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.

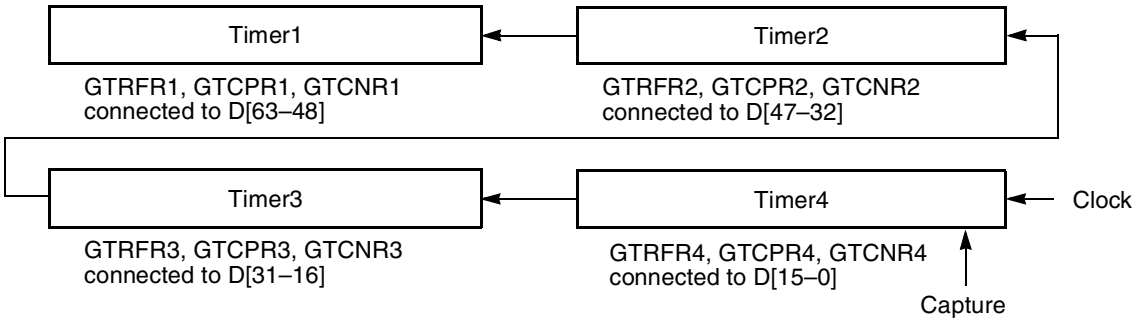


Figure 5-50. Timers Super-Cascaded Mode Block Diagram

5.7.7 Initialization/Application Information

5.7.7.1 Programming Guidelines

5.7.7.1.1 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 GTGCR and GTPSR is not initialized before the GTMDR. Only GTGCR[RST] can be modified at any time

- Clear GTEVR*n*[REF] and GTEVR*n*[CAP] by writing 1's 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 GTGCR*n*[STP] and to GTGCR*n*[RST] in order to initialize the appropriate timer's operation.

5.8 Power Management Control

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.

5.8.1 Modes of Operation

The device uses one or more of the following methods for power saving:

- Dynamic power management
- Shutting down unused blocks
- Software-controlled power-down states

5.8.2 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 PowerPC core are in low power state.	
		State Meaning	Asserted—The system and PowerPC core are in low power state. Negated—The system and PowerPC core are not in low power state.
		Timing	The timing between a quiesce request from the PowerPC core and the assertion of the external indication or between negation of the core's quiesce request and negation of the external indication depends on the current state of the internal system units and may vary accordingly.

5.8.3 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

Local Memory Offset (Hex)	Register	Access	Reset	Section/Page
0x00B00	Power management controller configuration register (PMCCR)	R/W	0x0000_0000	5.8.3.1/5-69
0x00B04	Power management controller event register (PMCER)	R/W	0x0000_0000	5.8.3.2/5-70
0x00B08	Power management controller mask register (PMCMR)	R/W	0x0000_0000	5.8.3.3/5-71
0x00B0C–0x00BFC	Reserved	—	—	—

5.8.3.1 Power Management Controller Configuration Register (PMCCR)

The power management controller configuration register (PMCCR), shown in Figure 5-51, controls whether only the PowerPC core will enter low power state upon quiesce request or additional parts of the device will also enter low power state.

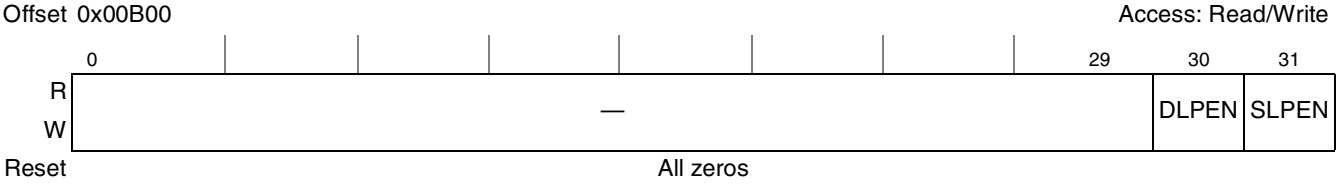


Figure 5-51. Power Management Controller Configuration Register

Table 5-5 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 will enter low power state when the rest of the system enters low power, according to SLPEN setting. DDR SDRAM will enter self-refresh mode (if enabled by DDR_SDRAM_CFG[SREN] memory controller register) and DDR clocks (MCK*n*) 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 will enter low power state when a quiesce request from the PowerPC core arrives. This bit is cleared when the device exits from low power state.

5.8.3.2 Power Management Controller Event Register (PMCER)

The power management controller event register (PMCER), shown in Figure 5-52, indicates with the PMCI bit that the power management controller has detected a wakeup event, that the system is not in idle state anymore, and that the device should exit low power state. If PMCMR[PMCIE] is set, the PMC interrupt request to the PowerPC core is driven.

Figure 5-52. Power Management Controller Event Register

Table 5-67 defines the bit fields of PMCER.

Table 5-67. PMCER Bit Settings

Bits	Name	Description
0–30	—	Reserved. Write has no effect, read returns 0.
31	PMCI	Power management controller interrupt. When set, indicates that the power management controller has detected that the system is not in idle state anymore, and that the device is required to exit low power state. If PMCMR[PMCIE] is set, the PMC interrupt request to the PowerPC core is driven, causing the PowerPC core to exit its low power state. PMCI can be cleared by writing a 1 to it (writing zero has no effect).

5.8.3.3 Power Management Controller Mask Register (PMCMR)

The power management controller mask register (PMCMR), shown in Figure 5-53, controls through the PMCIE bit whether the PMC interrupt request to the PowerPC core is enabled. The PMC interrupt request causes the PowerPC core to exit its low power state before any transaction on the system bus occurs.

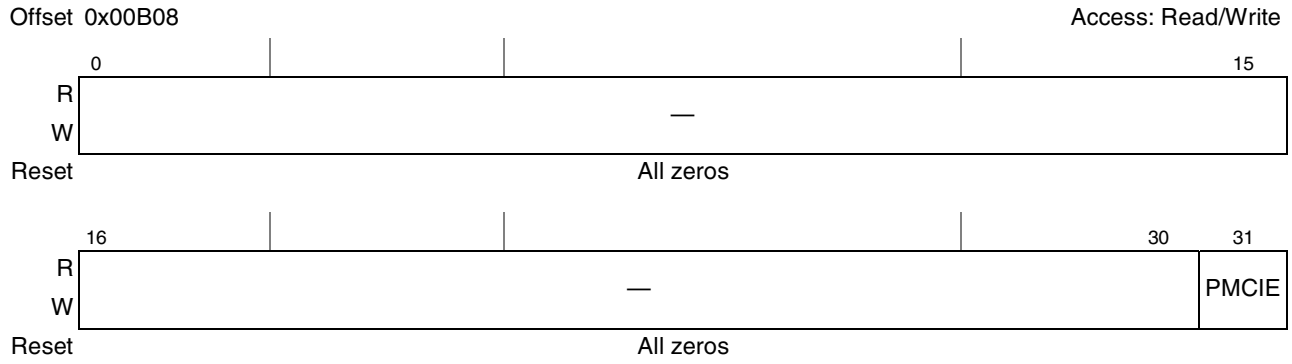


Figure 5-53. Power Management Controller Mask Register

Table 5-68 defines the bit fields of PMCMR.

Table 5-68. PMCMR Bit Settings

Bits	Name	Description
0–30	—	Reserved. Write has no effect, read returns 0.
31	PMCIE	Power management controller interrupt enable. 0 PMC interrupt request (PMCI) is disabled. 1 PMC interrupt request (PMCI) is enabled.

NOTE

The user is also required to enable the PMC interrupt in the programmable interrupt controller by setting SIMR_L[PMC].

5.8.4 Functional Description

The device has features to minimize power consumption at several levels. Dynamic power management locally minimizes power consumption when a block is idle. Software can also 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 PowerPC core can access the core's SPRs to put the device into doze, nap, or sleep power down states. Finally, software can access the PMCCR register to enable the device to go to low power state whenever the PowerPC core enters nap or sleep states. These power management features are described in further detail in this section.

5.8.4.1 Dynamic Power Management

Many blocks in the device can dynamically turn off clocks within the block when sections of the block are idle. This feature is always enabled and occurs automatically.

5.8.4.2 Shutting Down 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 PowerPC 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.

5.8.4.3 Software-Controlled Power-Down States

PowerPC 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 PowerPC core request to enter nap or sleep modes, it will also cause the system internal logic units to enter low power mode.

5.8.4.3.1 Entering Low Power States—Core-Only Mode

Entering Doze mode is controlled only by the e300 PowerPC core itself, and does not involve the power management controller or other blocks. For a more detailed description, see Table 7-1.

Entering Nap or Sleep modes occurs by writing to HID0 in the core, causing the core to make a quiesce request to the power management controller while PMCCR[SLPEN] is cleared. The core is immediately enabled to enter low power state, regardless of the system status. Note that since the core does not snoop the bus in this mode, it is the user's responsibility to keep the cache coherent. Other device peripheral and internal units continue to operate in full-on mode while the core is in low power state in this mode.

5.8.4.3.2 Entering Low Power States—Core and System Mode

Core and system mode is achieved when the core makes a quiesce request to the power management controller after PMCCR[SLPEN] is set. To preserve cache coherency and otherwise avoid loss of system state, the core's transition to low-power modes is coordinated with other functional blocks. 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] 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 will remain valid while the memory controller and its clocks are off. The DDR clocks are then disabled. 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 it's request to enter power down mode. Finally the $\overline{\text{QUIESCE}}$ output signal is asserted.

5.8.4.4 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 core has received an interrupt request.
- The power management controller has detected that the system is not idle and there are outstanding requests for 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.

5.8.4.4.1 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. For a detailed description, see Table 7-1.

Nap or Sleep modes are exited when the core has received an interrupt request, or according to the internal time base unit of the core (Nap mode only). The source of the interrupt can be an internal block or external signal. 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 rest of the system.

5.8.4.4.2 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, the TSEC receives an Ethernet frame, and requires to store it on the DDR SDRAM memory.

If the DDR SDRAM memory controller is in low power state (PMCCR[DxLPEN] was set when entering low power state), the power management controller initially enables the DDR SDRAM memory controller. DDR SDRAM clocks (MCK*n)* are enabled and the memory controller exit self-refresh and returns to auto-refresh mode.

The power management controller then enables other internal units and interrupts the PowerPC core. When all internal units, including the core, are ready, the power management controller enables the device to return to full-on state, negate the $\overline{\text{QUIESCE}}$ output, and clear PMCCR[SLPEN]. Outstanding requests for transactions are now granted to execute on the internal bus.

NOTE

Software is required to enable PMCI interrupt by setting PMCMR[PMCIE], otherwise exiting from low power state is not possible.

NOTE

It is the software’s responsibility to clear PMCER[PMCI].

5.8.5 Initialization/Application Information

5.8.5.1 Core Disable in Low Power Mode

If the device is required to operate with the core permanently disabled, the following steps must be taken:

1. Initialize the device with the core enable
2. Clear PMCCR[SLPEN] and disable the core time base unit by clearing SPCR[TBEN]. See Section 5.3.2.4, “System Priority and Configuration Register (SPCR),” for more information.
3. The e300 core enters low power state by accessing the HID0 register.
4. Set ACR[COREDIS] in the system arbiter register with an external master (that is, PCI). This steers all device interrupts to the $\overline{\text{INTA}}$ so the core cannot receive any interrupt requests.

NOTE

Make sure that the core cannot receive any interrupt requests during the time interval between setting HID0 and setting ACR[COREDIS]. This can be achieved if the rest of the system is idle (during the initialization).

By following this flow, the e300 core remains in low power state while the rest of the system is operational, and does not get out of this state as a result of any interrupt or time-based event.

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. 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.

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

Memory Offset (Hex)	Register	Access	Reset	Section/Page
0x00	Arbiter configuration register (ACR)	R/W	0x0000_0000/ 0x0010_0000[1]	6.2.1/6-2
0x04	Arbiter timers register (ATR)	R/W	0x00FF_00FF	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[2]	6.2.6/6-7
0x1C	Arbiter event address register (AEADR)	R	0x0000_0000[2]	6.2.7/6-9
0x20	Arbiter event response register (AERR)	R/W	0x0000_0000	6.2.8/6-10

[1] Reset value is determined from the core PLL configuration of the reset configuration word. See Chapter 4, "Reset, Clocking, and Initialization," for details.

[2] The registers AEATR and AEADR are affected only by the assertion of $\overline{\text{PORESET}}$

6.2.1 Arbiter Configuration Register (ACR)

Arbiter configuration register (ACR) defines the arbiter modes and parked master on the bus. Figure 6-1 shows the fields of ACR.

Offset 0x00 Access: User read/write

	0	6	7	8	12	13	15
R W	—		COREDIS	—		PIPE_DEP	
Reset	All zeros[1]						

	16	17	19	20	21	23	24	25	26	27	28	31
R W	—	PCI_RPTCNT		—	RPTCNT		—	—	APARK		PARKM	
Reset	All zeros[1]											

[1] 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 it's 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	—	Write reserved, read = 0
17–19	PCI_RPTCNT	PCI repeat count. Specifies the maximum number of consecutive transactions, that PCI master can perform, using $\overline{\text{REPEAT}}$ request mode. 000 One consecutive transaction ($\overline{\text{REPEAT}}$ request mode disable) 001 Two consecutive transactions 010 Three consecutive transactions 011 Four consecutive transactions 100 Five consecutive transactions 101 Six consecutive transactions 110 Seven consecutive transactions 111 Eight consecutive transactions
20	—	Write reserved, read = 0
21–23	RPTCNT	Repeat count. Specifies the maximum number of consecutive transactions, that any master (except PCI) 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 **Note:** It is recommended not to program this field for more than four consecutive transactions.
24–25	—	Write reserved, read = 0

Table 6-2. ACR Field Descriptions (continued)

Bits	Name	Description
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 $\overline{BG}$ to any master, if no $\overline{BR}$ is present. 11 Reserved
28–31	PARKM	Parking master. 0000 e300 core 0001 PCI1, PCI2, DMA 0010 TSEC1 0011 TSEC2 0100 USB 0101 Encryption core 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.

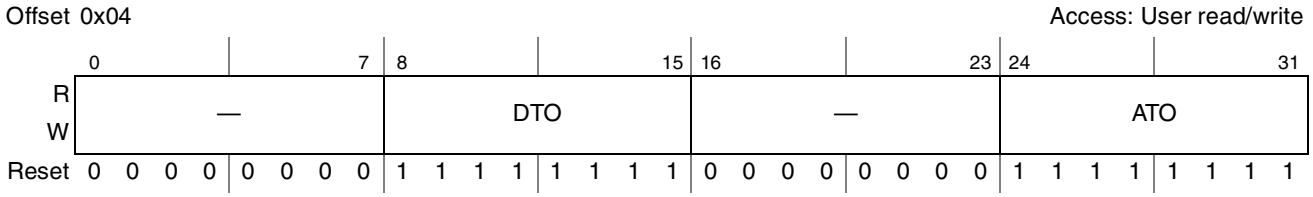


Figure 6-2. Arbiter Timers Register (ATR)

Table 6-3 describes ATR fields.

Table 6-3. ATR Field Descriptions

Bits	Name	Description
0–7	—	Write reserved, read = 0
8–15	DTO	Data time out. Specifies the time-out period for the data tenure. The granularity of this field is 64 bus clocks. The maximum value is 16320 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*64. 0x00 Reserved 0x01 64 clock cycles 0x10 128 clock cycles 0x11 192 clock cycles ... 0xFF 16320 clock cycles

Table 6-3. ATR Field Descriptions (continued)

Bits	Name	Description
16–23	—	Write reserved, read = 0
24–31	ATO	Address time out. Specifies the time-out period for the address tenure. The granularity of this field is 64 bus clocks. Maximum value is 16320 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*64. 0x00 Reserved 0x01 64 clock cycles 0x10 128 clock cycles 0x11 192 clock cycles ... 0xFF 16320 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

	0 – 25	26	27	28	29	30	31
R	—	ETEA	RES	ECW	AO	DTO	ATO
W		w1c	w1c	w1c	w1c	w1c	w1c
Reset	All zeros						

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	ETEA	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.
28	ECW	External control word transfer type. Reports on transaction with external control word transfer type. See Section 6.3.2.6, “Illegal (eciwx/ecowx) Transaction Type,” 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 Section 6.3.2.4, “Address Only Transaction Type,” for more information. 0 No transaction with address only transfer type occurred. 1 Transaction with address only transfer type occurred.

Table 6-4. AER Field Descriptions (continued)

Bits	Name	Description
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)

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

	0	25	26	27	28	29	30	31
R W	—		ETEA	RES	ECW	AO	DTO	ATO
Reset	All zeros							

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.
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)

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

	0	25	26	27	28	29	30	31
R W	—		ETEA	RES	ECW	AO	DTO	ATO
Reset	All zeros							

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.
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)

Arbiter event attributes register (AEATR) reports the type of transaction that causes error, which 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. Refer to Section 6.4.2, “Error Handling Sequence” for more information.

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 11*c* Reserved
8–10	—	Write reserved, read = 0
11–15	MSTR_ID	Master Id. 00000 e300 core data transaction 00001 Reserved 00010 e300 core instruction fetch 00011 Reserved 00100 TSEC1 00101 TSEC2 00110 USB MPH 00111 USB DR 01000 Encryption core 01001 I2C (boot sequencer) 01010 JTAG 01011 Reserved 01100 Reserved 01101 PCI1 01110 PCI2 01111 DMA 10000- 11111 Reserved **Note:** Master Id reflects the source of transaction and is used for debug purpose.
16–19	—	Write reserved, read = 0

Table 6-7. AEATR Field Descriptions (continued)

Bits	Name	Description
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: 001 1 byte 010 2 bytes 011 3 bytes 100 4 bytes 101 5 bytes 110 6 bytes 111 7 bytes 000 8 bytes TBST = 0: 000 16 bytes 001 24 bytes 010 32 bytes 011–111 Reserved
24–26	—	Write reserved, read = 0
27–31	TTYPE	Transfer type. 00000 Address-only 00001 Address-only 00010 Single-beat or burst write 00011 Reserved 00100 Address-only 00101 Reserved 00110 Burst write 00111 Reserved 0100x Address-only 0101x Single-beat or burst read 0110x Address-only 01110 Burst read 01111 Reserved 10000 Address-only 1XX01 Reserved 10010 Single-beat write 1XX11 Reserved 10100 **ecowx**—Illegal single-beat write 10110 Reserved 11000 Address-only 11010 Single-beat or burst read 11100 **eciwx**—Illegal single-beat read 11110 Burst read

6.2.7 Arbiter Event Address Register (AEADR)

Arbiter event address register (AEADR) reports the address of transaction that causes the error, which 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 effected by soft or hard reset, software can read this register and determine the cause of the bus failure, even if the bus failure had caused a deadlock situation. Refer to Section 6.4.2, “Error Handling Sequence,” for more information.

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 Section 6.2.6, "Arbiter Event Attributes Register (AEATR)," for more information.

6.2.8 Arbiter Event Response Register (AERR)

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.

Offset 0x20 Access: User read/write

	0	25	26	27	28	29	30	31
R	—		ETEA	RES	ECW	AO	DTO	ATO
W								
Reset	All zeros							

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.
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 arbiter functionality: 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.

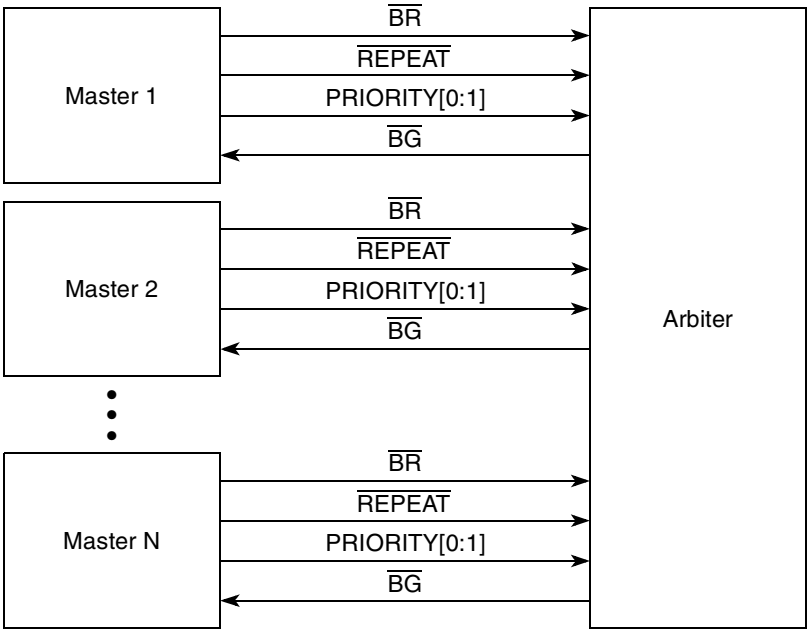


Figure 6-9. Address Bus Arbitration

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 REPEAT & 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 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 PRIORITY[0:1]

Whenever a master asserts its bus request to acquire address bus ownership, it can drive its 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 1/2 of the bus bandwidth
- M4 and M5 each gets 1/6 of the bus bandwidth
- M0 and M3 each gets 1/18 of the bus bandwidth
- M1 and M2 each gets 1/36 of the bus bandwidth

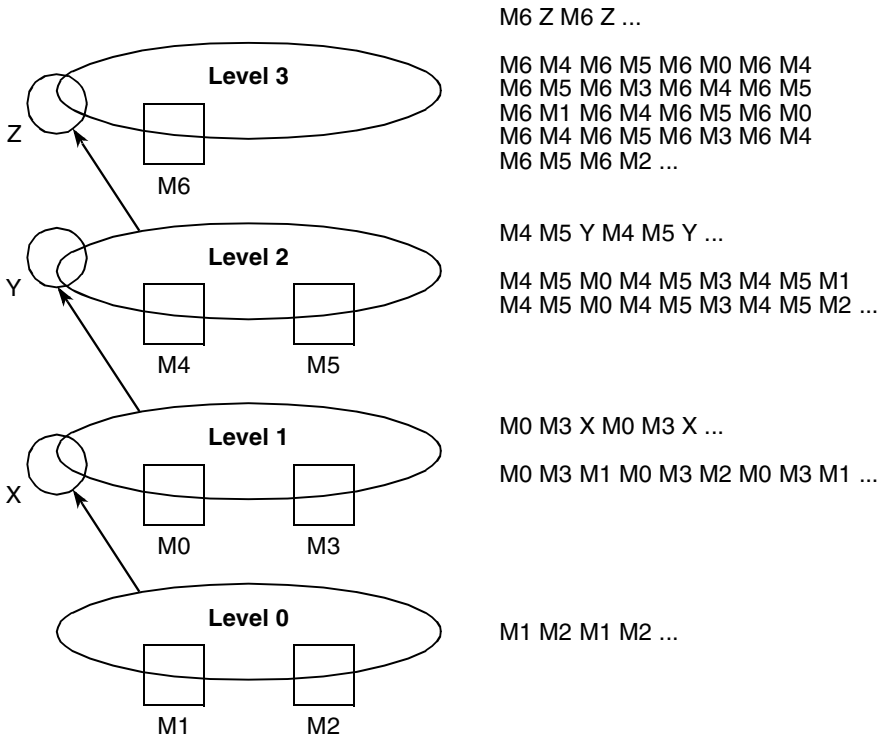


Figure 6-10. An Example of Priority-Based Arbitration Algorithm

NOTE

See each bus master's chapter and Section 5.3.2.4, "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, arbiter ignores the $\overline{\text{REPEAT}}$ signal and falls back to the regular arbitration scheme. PCI master has a dedicated repeat counter as it might need more repeated transactions before accepting read requests. PCI ordering rules require that the PCI bridge should empty all queued write operations before any new read operation can begin. See Section 3.2.5, "Transaction Ordering and Posting," of the *PCI Local Bus Specifications Rev 2.2* for more information.

See Section 6.2.1, "Arbiter Configuration Register (ACR)," for more details about programming ACR[RPTCNT] and ACR[PCI_RPTCNT].

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 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, which 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.

The transaction with an address only transfer type, the arbiter performs as follows:

1. Ends the address tenure by asserting $\overline{\text{AACK}}$.
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

The transaction with a reserved transfer type, the arbiter performs as follows:

1. Ends the address tenure by asserting $\overline{\text{AACK}}$.
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 (**ecowx**)
11100	External control word read (**eciwx**)

The transaction with an illegal (eciwx, ecowx) transfer type, the arbiter performs as follows:

1. Ends the address tenure by asserting $\overline{\text{AACK}}$.
2. Starts data tenure and ends data tenure by asserting $\overline{\text{TEA}}$.
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.

See 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),” and Section 6.2.8, “Arbiter Event Response Register (AERR),” for more information.

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, PCI 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 e300 processor core, a derivative of the original MPC603e PowerPC microprocessor design. The e300 core is an implementation of the PowerPC microprocessor family. Throughout this document, the terms ‘e300 core’, ‘core’, and ‘processor’ are used interchangeably. The terms ‘e300c1’ are used when describing an implementation-specific feature while the term ‘e300’ is used when describing a feature that pertains to the family of processors. The MPC8349EA uses an e300c1 core.

7.1 Overview

This section describes the details of the e300 core, provides a block diagram showing the major functional units (see Section 3.1.2, “Endian Modes and Byte Ordering”), and briefly describes how these units interact. All differences between the e300 and previous PowerPC implementations derived from the MPC603e processor are noted. For additional information, please refer to the *e300 PowerPC Core 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 e300 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 e300c1 core integrates five execution units: 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 ability to execute five 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. On the e300c1 core, the FPU is pipelined so a single-precision multiply-add instruction can be issued and completed every clock cycle. The e300c1 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 e300c1 core.

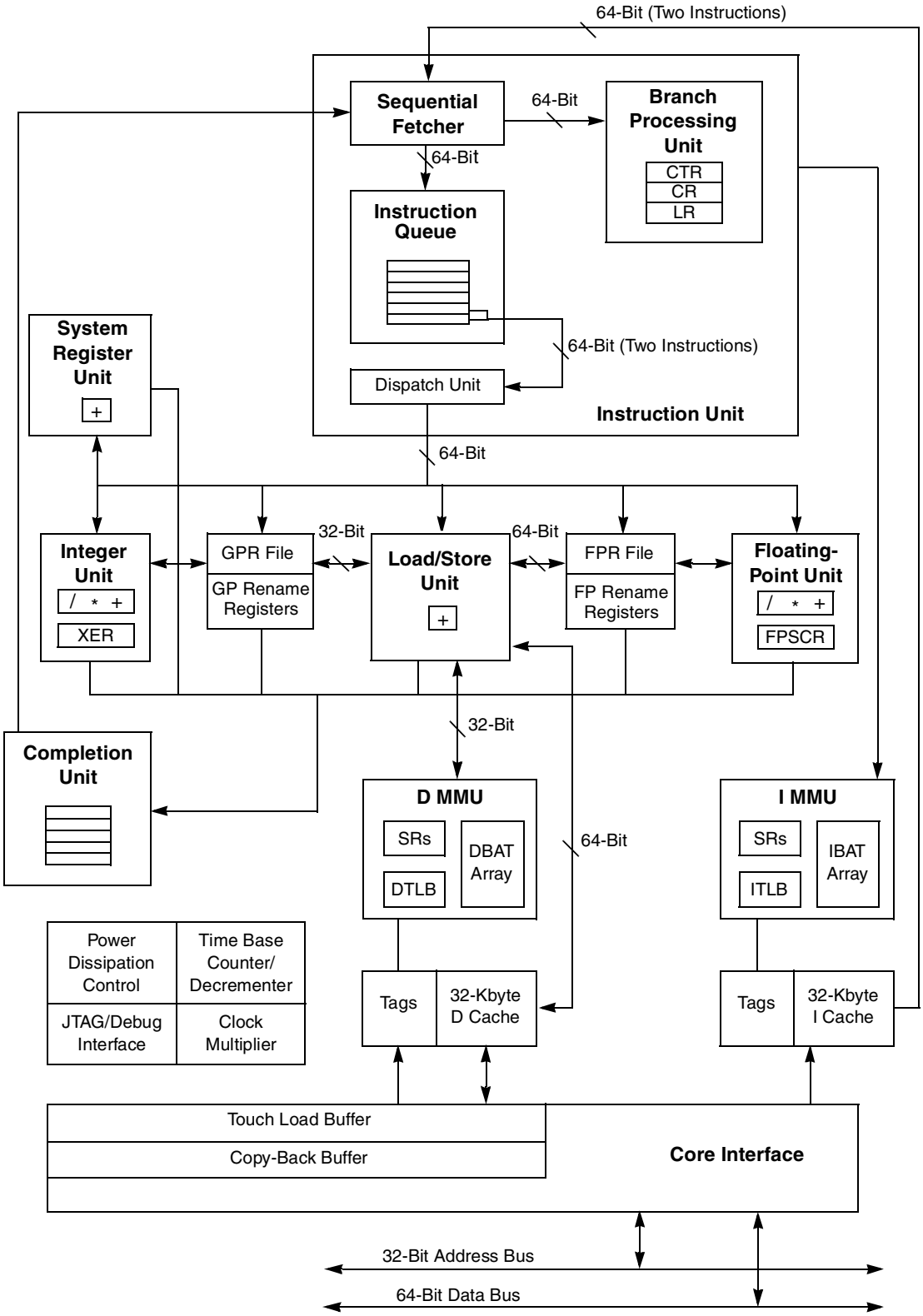


Figure 7-1. e300c1 Core Block Diagram

The e300c1 core provides independent, on-chip, 32-Kbyte, eight-way, set-associative, physically-addressed caches for instructions and data, and 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 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 e300 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. The core interface protocol allows multiple masters to compete for system resources through a central external arbiter. 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 MPC8349EA. The e300 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. See Section 7.3.8, “Debug Features,” for more detail. Additional SPRG interrupt handling registers are provided for enhancing flexibility for the operating system.

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-precision and double-precision operations

- Independent execution units and two register files
  - Branch processing unit (BPU) featuring static branch prediction
  - A 32-bit integer unit (IU) in the e300c1
  - Fully IEEE® 754™–compliant FPU 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
  - 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
  - 32-Kbyte data cache and 32-Kbyte instruction cache with parity—eight-way, set-associative, physically addressed, PLRU replacement algorithm.
  - 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
  - A64-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 *MPC8349EA Integrated Processor Hardware Specifications*

- 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
  - 32-Kbyte, eight-way, set-associative instruction and data cache;
  - 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 7 of 8 ways on e300c1
  - New **icbt** instruction supports initialization of instruction cache
  - Data cache supports four-state MESI coherency protocol (not implemented on MPC8349EA)
  - 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.
- 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 $\overline{ext_halt}$

Figure 7-1, provide block diagrams of the e300 cores 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, 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 , the e300 core instruction unit, containing 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 , 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 7.3.6, "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***x*) instruction. The CTR contains the branch target address for the Branch Conditional to Count Register (**bcctr***x*) 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 e300 core writes the contents of the rename registers to the appropriate GPR when integer instructions are retired by the completion unit.

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 e300c1 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 e300c1 core writes the contents of the rename registers to the appropriate FPR when floating-point instructions are retired by the completion unit.

The e300c1 cores support all IEEE 754 floating-point data types (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 e300 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 e300 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 **tlbld**) 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.

See Section 7.3.5.2, “Implementation-Specific Memory Management,” for more information about memory management for the core.

7.1.5.2 Cache Units

The e300c1 core provides independent, 32-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 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 will request entry via a *qreq* signal and will only enter another power mode after an acknowledge (*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 will automatically enter 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 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 7.3.8, "Debug Features," for more information.

The *stopped* signal allows observation of the internal clock state, and *ext_halt* can be used to force the core into a halted state. See Chapter 5, "Interrupts and Exceptions," for more information on test signals.

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).

7.2 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.

7.3 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 7.3.1, "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 7.3.2, “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 7.3.3, “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 7.3.4, “Interrupt Model,” describes the interrupt model of the OEA and the differences in the core interrupt model.
- Section 7.3.5, “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 7.3.6, “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.”

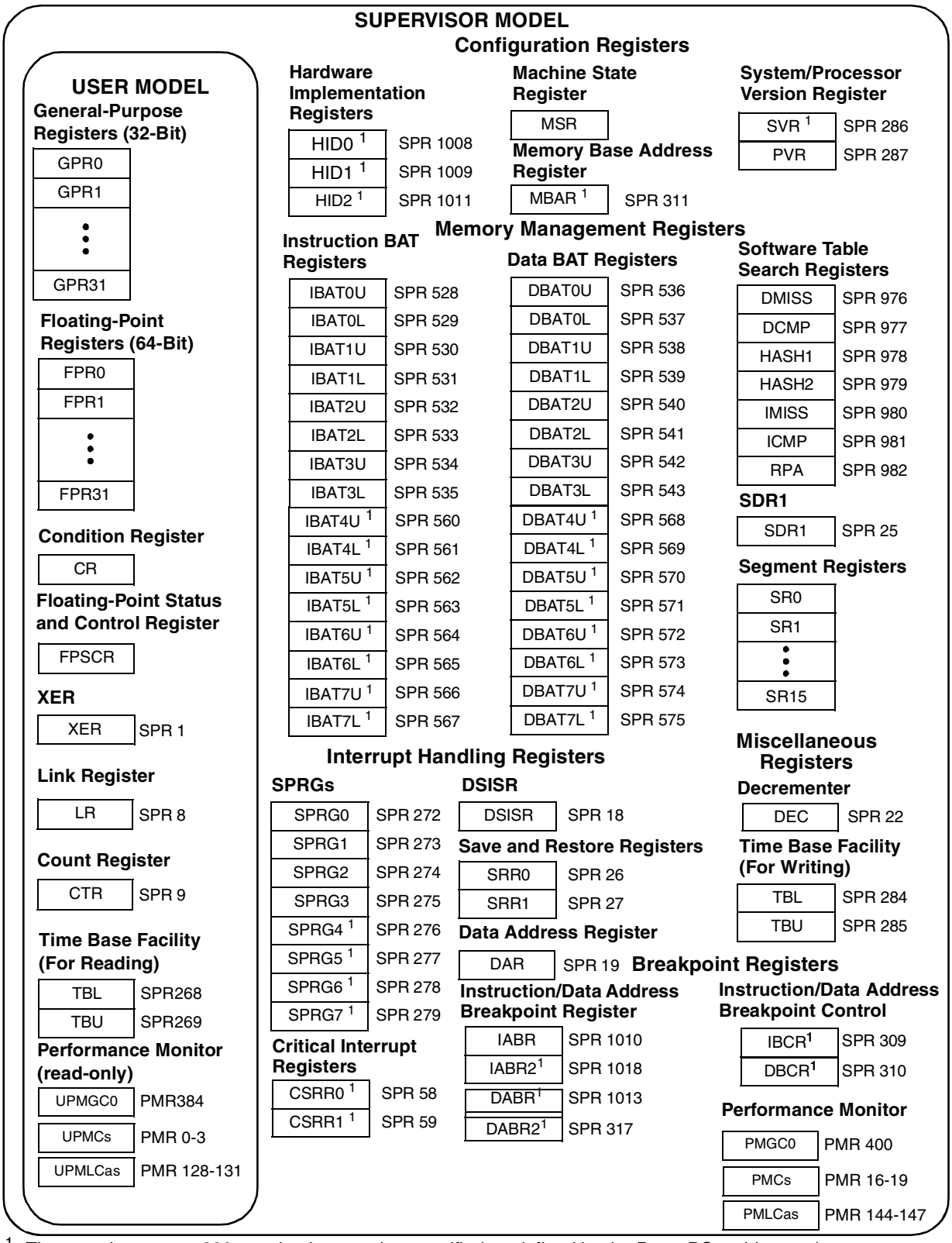
7.3.1 Register Model

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.



1 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.

7.3.1.1 UISA Registers

UISA registers are user-level registers that include the following.

7.3.1.1.1 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.

7.3.1.1.2 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.

7.3.1.1.3 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.

7.3.1.1.4 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.

7.3.1.1.5 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.

7.3.1.2 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.

7.3.1.3 OEA Registers

OEA registers are supervisor-level registers that include the following.

7.3.1.3.1 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 interrupt is taken in the e300 core when the *cint* signal is asserted and MSR[CE] is set. The e300 core implements the MSR as a 32-bit register.

Table 7-4 shows the bit definitions for MSR.

Table 7-1. MSR Bit Descriptions

Bits	Name	Description
0[1]	—	Reserved. Full function.
1–4[1]	—	Reserved. Partial function.
5–9[1]	—	Reserved. Full function.
10–12[1]	—	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 **mtmsr** instruction only. Also, when altering the POW bit, software may alter only this bit in the MSR and no others. The **mtmsr** instruction must be followed by a context-synchronizing instruction. See Chapter 9, "Power Management," for more information.
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 **rfi** 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.
16	EE	External interrupt enable 0 The processor ignores external interrupts, system management interrupts, and decrementer interrupts. 1 The processor is enabled to take an external interrupt, system management interrupt, or decrementer interrupt.

Table 7-1. MSR Bit Descriptions (continued)

Bits	Name	Description
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 For e300: 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 (see Table 5-9)
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 (see Table 5-9)
24	CE	Critical interrupt enable 0 Critical interrupts disabled 1 Critical interrupts enabled; critical interrupt and **rfci** instruction enabled The critical interrupt is an asynchronous implementation-specific interrupt. The critical interrupt vector offset is 0x00A00. The **rfci** 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, *nnnnn* is the offset of the interrupt. See Table 5-2. 0 Interrupts are vectored to the physical address 0x000*n_nnnn* 1 Interrupts are vectored to the physical address 0xFFF*n_nnnn*
26	IR	Instruction address translation 0 Instruction address translation is disabled 1 Instruction address translation is enabled See Chapter 6, “Memory Management.”
27	DR	Data address translation 0 Data address translation is disabled 1 Data address translation is enabled See Chapter 6, “Memory Management.”
28–29[1]	—	Reserved. Full function.

Table 7-1. 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. See Section 3.1.2, "Endian Modes and Byte Ordering," for a description of the core operating in true little-endian mode.

[1] All reserved bits should be set to zero for future compatibility.

7.3.1.3.2 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.

7.3.1.3.3 Supervisor-Level SPRs

The e300 core, like the G2_LE core, has additional supervisor-level SPRs, which are shown in Figure 7-2. 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.

Supervisor-level SPRs include the following:

- 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 decrementer register (DEC) is a 32-bit decrementing counter that provides a mechanism for causing a decrementer 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 7-7 for the version and revision level of the PVR for the e300 processor core.
- Block address translation (BAT) arrays—The PowerPC architecture defines 16 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, 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 7-2 shows the bit definitions for HID0.

Table 7-2. e300 HID0 Bit Descriptions

Bits	Name	Function
0	EMCP	Enable *mcp*. The purpose of this bit is to mask out machine check interrupts caused by assertion of *mcp*, similar to how MSR[EE] can mask external interrupts. 0 Masks *mcp*. Asserting *mcp* does not generate a machine check interrupt or a checkstop. 1 Asserting *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 *ap_in[0:3]* and *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 *dpe* 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 *hreset* to configure *clk_out.*
5	—	Reserved, should be cleared
6	ECLK	*clk_out* output enable. Used in conjunction with HID0[SBCLK] and the *hreset* signal to configure *clk_out.*
7	PAR	Disable precharge of *artry_out* 0 Precharge of *artry_out* enabled 1 Alters bus protocol slightly by preventing the processor from driving *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. *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, *qack,* is asserted back to the processor. Once *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*.
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.

Table 7-2. e300 HID0 Bit Descriptions (continued)

Bits	Name	Function
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 **isync** 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 **sync** 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 **mtspr** operations.

Table 7-2. 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 **mtspr** 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
25–26	—	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 **dcbi**, **dcbf**, and **dcbst**. 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 **dcbt** and **dcbtst** instructions are enabled 1 The **dcbt** and **dcbtst** instructions are no-oped internal to the e300 core

Table 7-3 shows the bit definitions for HID1

Table 7-3. 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 7-4 shows the bit definitions for HID2

Table 7-4. 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 *tle* signal at the negation of $\overline{hreset}$. 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
11–12	—	Reserved for e300c1, should be cleared.

Table 7-4. 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 and adjusted (weighted) LRU such that they always select and replace the lowest unlocked way in the data cache.
12	—	Reserved for e300c1, should be cleared.
13	HBE	High BAT enable. Regardless of the setting of HID2[HBE], these BATs are accessible by **mfspr** and **mtspr**. 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 3 locked 101 = way 0 through way 4 locked 110 = way 0 through way 5 locked 111 = way 0 through way 6 locked Setting HID0[ILOCK] will lock 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 3 locked 101 = way 0 through way 4 locked 110 = way 0 through way 5 locked 111 = way 0 through way 6 locked Setting HID0[DLOCK] will lock all ways
27–31	—	Reserved, should be cleared.

7.3.2 Instruction Set and Addressing Modes

The following sections describe the PowerPC instruction set and addressing modes in general.

7.3.2.1 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—These include computational and logical instructions.
  - Integer arithmetic instructions
  - Integer compare instructions
  - Integer logical instructions
  - Integer rotate and shift instructions
- Floating-point instructions—These include floating-point computational instructions, as well as 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—These include 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 **stwcx.** instructions)
- Flow control instructions—These include 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—These instructions are used for synchronizing memory accesses and management of caches, TLBs, and the segment registers.
  - Move to/from SPR instructions
  - Move to/from MSR
  - Synchronize
  - Instruction synchronize

- Memory control instructions—These instructions 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 which are 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.

7.3.2.2 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 which is added to support critical interrupts (also supported on the G2_LE). This is a supervisor-level, context synchronizing instruction.
  - Return from Critical Interrupt (**rfci**)

- The core implements the following instruction which is added to support easy start-up initialization or reloading of the instruction cache.
  - Instruction Cache Block Touch (**icbt**)
- The core provides the following perfmormance monitor instructions:
  - Move to Performance Monitor Register (**mtpmr**)
  - Move from Performance Monitor Register (**mfpmr**)

7.3.3 Cache Implementation

The following sections describe the general cache characteristics as implemented in the PowerPC architecture and the core implementation.

7.3.3.1 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.

7.3.3.2 Implementation-Specific Cache Implementation

The e300c1 core have two 32-Kbyte, eight-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 8 blocks each. 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 also consists of 128 sets of 8 blocks, and 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 organization of the instruction cache is very similar to the data cache shown in Figure 7-3.

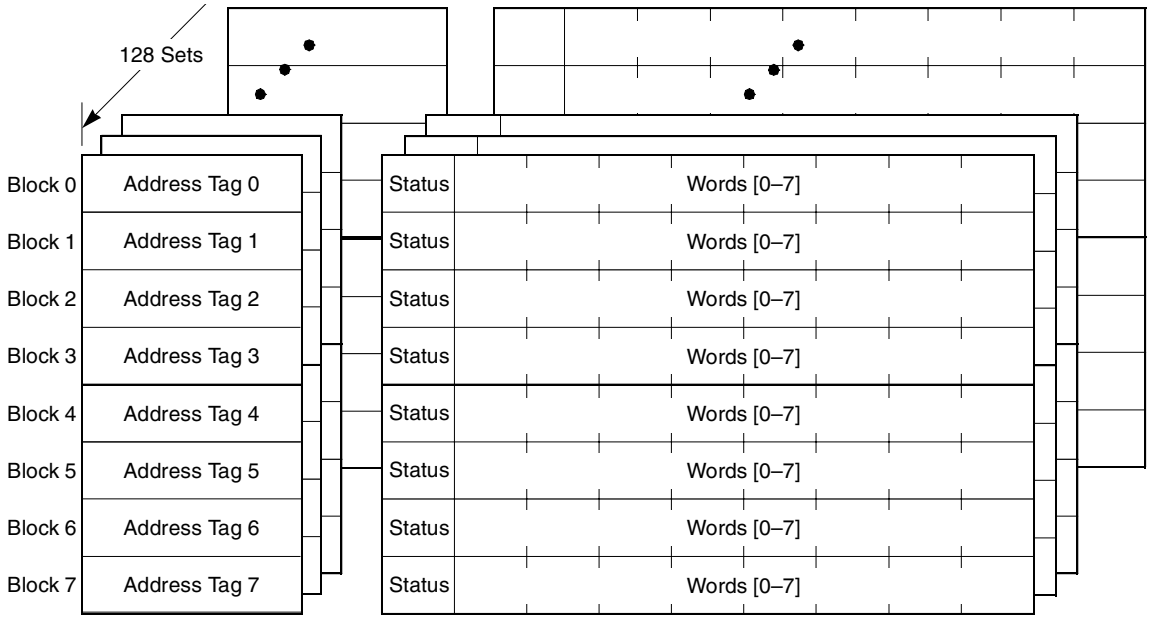


Figure 7-3. e300c1 Data Cache Organization

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 MESI is supported by the e300 core, it is not implemented on MPC8349EA. 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 MESI is supported by the e300 core, it is not implemented on MPC8349EA.
- 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.

7.3.3.3 Instruction and Data Cache Way-Locking

The e300 core implements instruction and data cache way-locking, which guarantees that certain memory accesses will hit in the cache. This provides deterministic access times for those accesses. See Chapter 4, "Instruction and Data Cache Operation," for more information.

7.3.4 Interrupt Model

This section describes the PowerPC interrupt model and the e300 core implementation specifically.

7.3.4.1 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 decrementer interrupt, have only a single exception condition. Exceptions and the interrupts they cause are described in Chapter 5, "Interrupts and Exceptions." 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, interrupt handlers 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 will complete 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 decrementer 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].

7.3.4.2 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 7-5. The interrupts are listed in Table 5-3 in order of highest to lowest priority.

Table 7-5. 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 7-5 define categories of interrupts that the core handles uniquely. Note that Table 7-5 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 7-6.

Table 7-6. Exceptions and Interrupts

Interrupt Type	Vector Offset (hex)	Exception Conditions
Reserved	00000	—
System reset	00100	Caused by the assertion of either $\overline{sreset}$ or $\overline{hreset}$.
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. See Table 5-14 for more information.
DSI	00300	Determined by the bit settings in the DSISR, listed as follows: 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 4 Set if a memory access is not permitted by the page or DBAT protection mechanism; otherwise cleared 6 Set for a store operation and cleared for a load operation 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: • Write breakpoints enabled when DABR[30] is set • Read breakpoints enabled when DABR[31] is set
ISI	00400	Caused when an instruction fetch cannot be performed for any of the following reasons: • 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. • 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.
External interrupt	00500	Caused when MSR[EE] = 1 and the $\overline{int}$ signal is asserted.

Table 7-6. Exceptions and Interrupts (continued)

Interrupt Type	Vector Offset (hex)	Exception Conditions
Alignment	00600	Caused when the core cannot perform a memory access for any of the reasons described below: • The operand of a floating-point load or store instruction is not word-aligned. • The operands of **lmw**, **stmw**, **lwarx**, and **stwcx.** instructions are not aligned. • The instruction is **lswi**, **lswx**, **stswi**, **stswx**, and the core is in little-endian mode. Note that PowerPC little-endian mode is not supported on the e300 core. • The operand of **dcbz** is in memory that is write-through-required or caching-inhibited.
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. • 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. • 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). • 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 **mtspr** or **mfspr** 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. • Trap—A trap type program interrupt is generated when any of the conditions specified in a trap instruction are met.
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[31] 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 (**sc**) 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.
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.
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.

Table 7-6. Exceptions and Interrupts (continued)

Interrupt Type	Vector Offset (hex)	Exception Conditions
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	—

7.3.5 Memory Management

The following sections describe the memory management features of the PowerPC architecture and the e300 core implementation, respectively.

7.3.5.1 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.

7.3.5.2 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.

7.3.6 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 five independent execution units, one each for integer instructions, floating-point instructions, branch instructions, load/store instructions, and system register instructions. 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. Refer to Chapter 7, “Instruction Timing,” for more information.

Because the PowerPC architecture can be applied to such a wide variety of implementations, instruction timing among processor cores varies accordingly.

7.3.7 Core Interface

The core interface is specific for each processor core implementation.

The MPC8349EA contains an internal coherent system bus (CSB) that interfaces the processor core to the peripheral logic. This internal bus is very similar in function to the external 60x bus interface on the MPC603e. In the case of the MPC8349EA, 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.

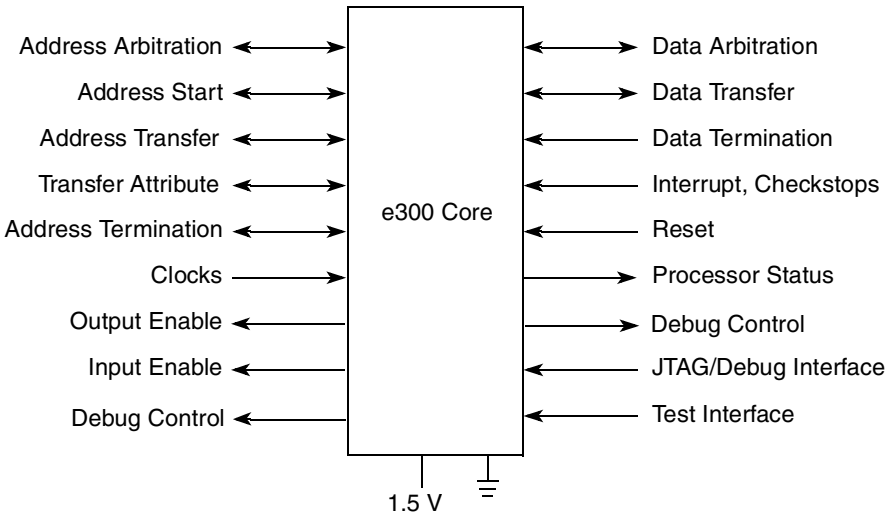


Figure 7-4. Core Interface

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.

7.3.7.1 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.

7.3.7.2 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, 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 (**IEEE** 1149.1–compliant) 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 (*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 enable signal, and the *tlbisync* 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.

NOTE

A bar over a signal name indicates that the signal is active low—for example, *artry_in* (address retry) and *ts_in* (transfer start). 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 *ap_in[0:3]* (address bus parity signals) and *tt_in[0:4]* (transfer type signals) are referred to as asserted when they are high and negated when they are low.

7.3.8 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 **mfspr** instructions. The e300 provides the following additional feature in the JTAG/debug interface: Inclusion of breakpoint status and control pins: *stopped* and *ext_halt.*

7.3.8.1 Breakpoint Signaling

The breakpoint signaling provided on the e300 core allows observability of breakpoint matches external to the core. The *iabr, iabr2, dabr,* and *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 *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 *iabr, iabr2* and *dabr, dabr2* reflect their respective breakpoints.
- When the DBCR and IBCR are configured for AND combinational signal type, only the *iabr2* and *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 either a softstop or a hardstop, conditional upon the HARDSTOP condition being set through the JTAG/debug interface

The breakpoint signaling conditions are described in Chapter 10, “Debug Features.”

7.4 Differences Between Cores

The e300c1 core has similar functionality to the G2_LE core. Table 7-7 describes the differences between the G2_LE and the e300c1.

Table 7-7. Differences Between e300c1 and G2_LE Cores

e300c1 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 e300c1 core version number is 0x8083, and the revision level starts at 0x0010 and changes for each revision of the core. The G2_LE core version number is 0x8082, and the revision level starts at 0x1010 and changes for each revision of the core.
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.
32-Kbyte, eight-way, set-associative, instruction and data caches	16-Kbyte, four-way, set-associative, instruction and data caches	The e300c1 core doubles the size (and doubles the number of ways) of the L1 instruction and data caches
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. Not implemented on MPC8349EA.
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 7-7. Differences Between e300c1 and G2_LE Cores (continued)

e300c1 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.
icbt 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.
1-1/2-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 will not be supported in the e300 core, although true little-endian will be fully supported.
Data retry mode removed	Data retry mode available	$\overline{drtry}$ and *drtrymode* will no longer be supported on the e300 and future versions.
External control instructions removed	External control instructions available	The **eciwx** and **ecowx** instruction pair will not be 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 *redpinmode* will not be 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 programming model is similar to the interrupt controller of the MPC8260. 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 (DDR)
- Local bus memory controller (LBC)
- PCI1 and PCI2
- Four-channel DMA controller (DMA)
- Message unit (MU)
- Dual three-speed Ethernet controllers (TSEC1 and TSEC2)
- DUART communication module (DUART)
- USB 2.0 controller (USB DR and USB MPH)
- Security engine (SEC)
- System bus arbiter (SBA)
- Periodic interval timer (PIT)
- Real time clock timer (RTC ALR and RTC SEC)
- Eight global timers (GTM1–GTM8)
- Software watchdog timer (WDT)
- I^2C controllers (I^2C1 and I^2C2)
- SPI controller (SPI)
- Power management controller (PMC)
- Two general-purpose I/O controllers (GPIO1 and GPIO2)
- External pins ($\overline{\text{IRQ}}$[0:7])

The interrupt sources controlled by the IPIC unit cause exceptions in the processor core. The internal interrupt ($\overline{int}$) signal is the main interrupt output from the IPIC to the core and it causes the regular interrupt exception. The $\overline{cint}$ 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 ($\overline{IRQ}$[0:7]).

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 interrupt pending 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.

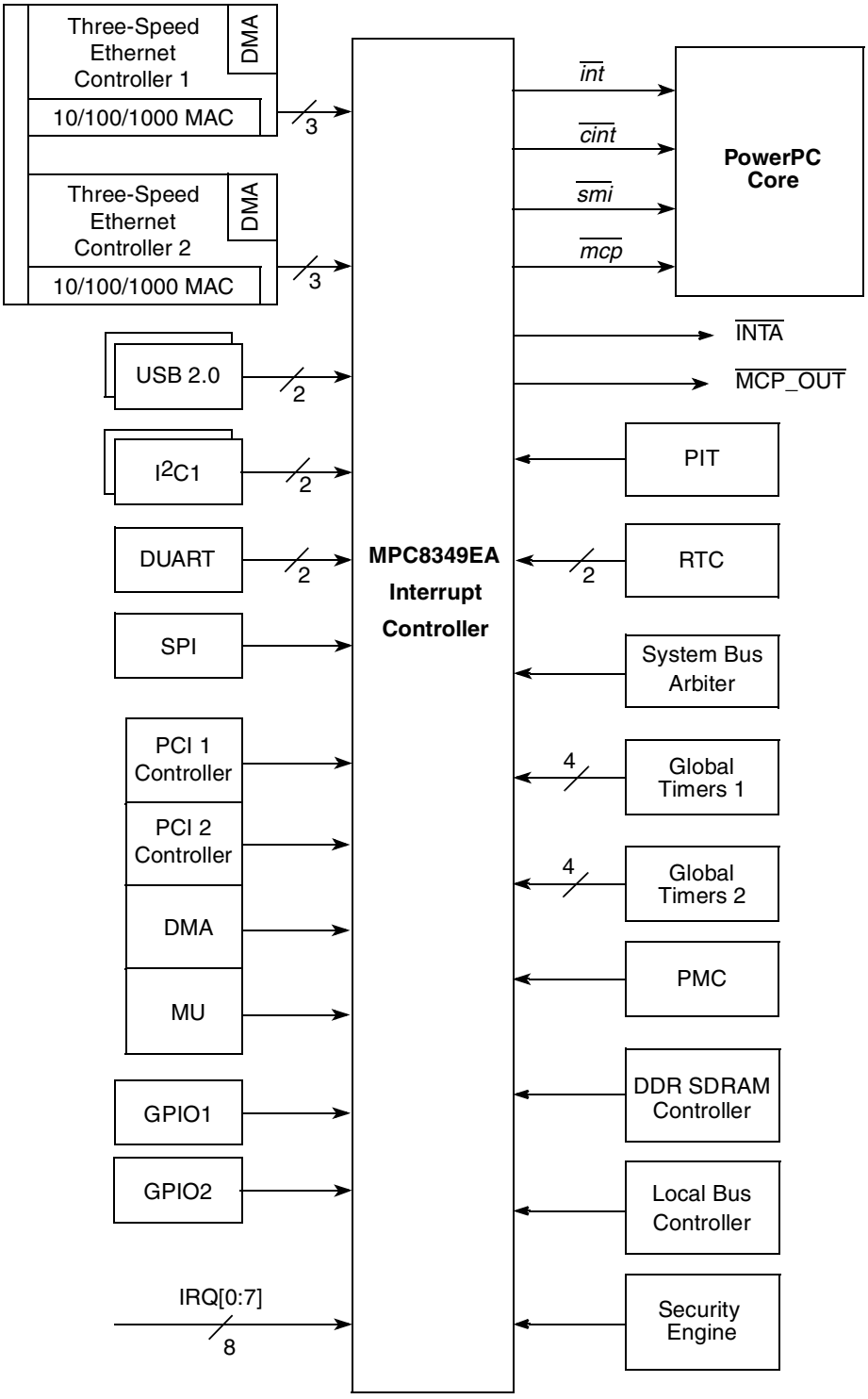


Figure 8-1. Interrupt Sources Block Diagram

The IPIC receives interruptsfrom the following types:

- External interrupt—off-chip signals ($\overline{\text{IRQ}}$[0:7])
- Internal interrupts—on-chip interrupt signals. Sources are listed in Table 8-7 and Table 8-9.
- 1 external and multiple internal interrupts—on-chip $\overline{\text{mcp}}$ interrupt signals. Sources are listed in Table 8-21.

The interrupt controller provides the ability to mask each interrupt source. Multiple events within the GPIO, SEC, or an SBA peripheral event are also maskable.

When the IPIC receives an internal or external interrupt, its configuration register is checked to determine if it should be routed off-chip (to the external $\overline{\text{INTA}}$) or serviced as a normal external interrupt by the processor core (through the $\overline{\textit{int}}$ signal). As a third alternative, if the incoming interrupt has been configured as a critical or system management interrupt, the IPIC completes the processing of the interrupt by asserting $\overline{\textit{cint}}$ or $\overline{\textit{smi}}$ to the core. The assertion of the $\overline{\textit{cint}}$ or $\overline{\textit{smi}}$ 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:

- Functional and programming compatibility with the MPC8260 interrupt controller
- Supports external and internal discrete vectorized interrupt sources
- Supports one external and 5 internal $\overline{\textit{mcp}}$ sources
- Programmable highest priority request (can be programmed to support a critical ($\overline{\textit{cint}}$) or system management ($\overline{\textit{smi}}$) interrupt 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
- Two 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 (including those from the PCI1 and PCI2 blocks) are routed to and from the IPIC; the interrupts are sent to the PowerPC core. The DMA controller can optionally (depending on the programming of the DMA registers) steer its interrupt to the PCI host through the $\overline{\text{INTA}}$ signal.

In this mode all machine check interrupts are gathered by the IPIC unit and sent to the PowerPC core. If the device performs as a PCI host, the interrupts of the other PCI agents should be connected to the implementation's IRQx signals and treated like normal external interrupts (sent to the core).

8.3.2 Core Disable Mode

In core disable mode, all internal interrupts (including those from the PCI1 and PCI2 blocks) are routed to and from the IPIC, the interrupts are then sent through the INTA signal to the PCI host CPU. Note that the core interrupt signal is masked. The user should use in this mode only the *int* output interrupt type (should not use *cint* or *smi* output interrupt types) to read an updated SIVCR. (See Section 8.5.7, "System Internal Interrupt Control Register (SICNR)," and Section 8.5.12, "System External Interrupt Control Register (SECNR).")

In this mode, machine check interrupts are driven either on INTA or on MCP_OUT as level-sensitive interrupts. SERCR[MCPR] (see Section 8.5.15, "System Error Control Register (SERCR)") controls which external signal is used.

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 8 distinct external interrupt request input signals (IRQ[0:7]) and one interrupt request output signal (INTA). The IPIC interface signals are defined in Table 8-1.

Table 8-1. IPIC Signal Properties

Name	Port	Function	I/O	Reset	Requires Pull Up
IRQ[0:7]	IRQ[0:7]	External interrupts	I	—	Yes
INTA	INTA	Interrupt request output	O	Z	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	
$\overline{\text{IRQ}}$[0:7]	I	Interrupt request 0–7. The sense (level or edge) of each of these signals is programmable. All of these inputs can be driven completely asynchronously.	
		State Meaning	Asserted—When an external interrupt request signal is asserted (according to the programmed polarity), 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.
		Timing	Assertion—All of these inputs can be asserted completely asynchronously. Negation—Interrupts programmed as level-sensitive must remain asserted until serviced.
$\overline{\text{INTA}}$	OD	Interrupt request out. Active-low, open drain. When the IPIC is programmed in core disable mode, this output reflects the raw interrupts generated by on-chip sources. See Section 8.3, “Modes of Operation,” for details.	
		State Meaning	Asserted—At least one interrupt is currently being signalled to the external system. Negated—Indicates no interrupt source currently routed to $\overline{\text{IRQ_OUT}}$.
		Timing	Because external interrupts are asynchronous with respect to the system clock, both assertion and negation of $\overline{\text{IRQ_OUT}}$ occur asynchronously with respect to the interrupt source. All timing given here is approximate. Assertion—Internal interrupt source: 3 system bus clock cycles after the interrupt occurs. External interrupt source: 4 cycles after the interrupt occurs. Negation—Follows interrupt source negation with the following delay: Internal interrupt: 3 system bus clock cycles. External interrupt: 4 cycles.
$\overline{\text{MCP_OUT}}$	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 $\overline{\textit{mcp}}$ interrupts generated by on-chip sources. See Section 8.3, “Modes of Operation.”	
		State Meaning	Asserted—At least one machine check interrupt is currently being signalled to the external system. Negated—Indicates no interrupt source currently routed to $\overline{\text{MCP_OUT}}$.
		Timing	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 2, “Memory Map.”

Table 8-3 shows the memory map of the IPIC unit.

Table 8-3. IPIC Register Address MapIRQ3

Offset	Register	Access	Reset Value	Section/ Page
0x00	System global interrupt configuration register (SICFR)	R/W	0x0000_0000	8.5.1/8-8
0x04	System regular interrupt vector register (SIVCR)	R	0x0000_0000	8.5.2/8-9
0x08	System internal interrupt pending register (SIPNR_H)	R	0x0000_0000	8.5.3/8-11
0x0C	System internal interrupt pending register (SIPNR_L)	R	0x0000_0000	8.5.3/8-11
0x10	System internal interrupt group A priority register (SIPRR_A)	R/W	0x0530_9770	8.5.4/8-14
0x14	Reserved	—	—	—
0x18	Reserved	—	—	—
0x1C	System internal interrupt group D priority register (SIPRR_D)	R/W	0x0530_9770	8.5.5/8-14
0x20	System internal interrupt mask register (SIMSR_H)	R/W	0x0000_0000	8.5.6/8-15
0x24	System internal interrupt mask register (SIMSR_L)	R/W	0x0000_0000	8.5.6/8-15
0x28	System internal interrupt control register (SICNR)	R/W	0x0000_0000	8.5.7/8-16
0x2C	System external interrupt pending register (SEPNR)	R/W	Special	8.5.8/8-18
0x30	System mixed interrupt group A priority register (SMPRR_A)	R/W	0x0530_9770	8.5.9/8-18
0x34	System mixed interrupt group B priority register (SMPRR_B)	R/W	0x0530_9770	8.5.10/8-19
0x38	System external interrupt mask register (SEMSR)	R/W	0x0000_0000	8.5.11/8-20
0x3C	System external interrupt control register (SECNR)	R/W	0x0000_0000	8.5.12/8-21
0x40	System error status register (SERSR)	R/W	0x0000_0000	8.5.13/8-22
0x44	System error mask register (SERMR)	R/W	0xFF00_0000	8.5.14/8-23
0x48	System error control register (SERCR)	R/W	0x0000_0000	8.5.15/8-24
0x4C–0x4F	Reserved	—	—	—
0x50	System internal interrupt force register (SIFCR_H)	R/W	0x0000_0000	8.5.16/8-25
0x54	System internal interrupt force register (SIFCR_L)	R/W	0x0000_0000	8.5.16/8-25
0x58	System external interrupt force register (SEFCR)	R/W	0x0000_0000	8.5.17/8-26
0x5C	System error force register (SERFR)	R/W	0x0000_0000	8.5.18/8-26
0x60	System critical interrupt vector register (SCVCR)	R	0x0000_0000	8.5.19/8-27
0x64	System management interrupt vector register (SMVCR)	R	0x0000_0000	8.5.20/8-27
0x68–0xFF	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. See Table 8-4 for more information.

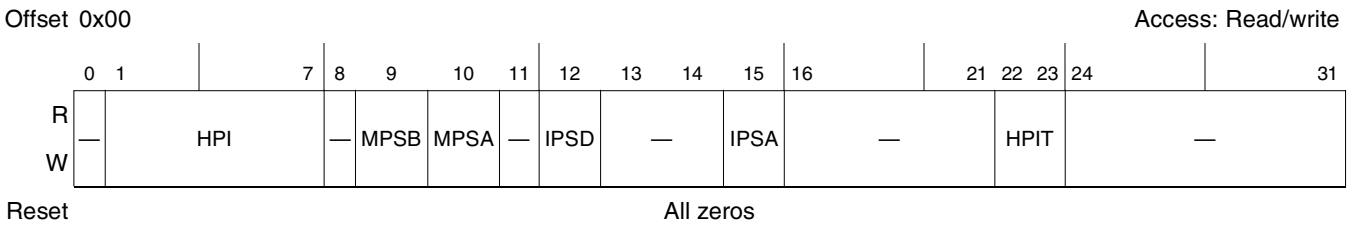


Figure 8-2. System Global Interrupt Configuration Register (SICFR)

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 8-31). 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.
13–14	—	Write ignored, read = 0
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

Table 8-4. SICFR Field Descriptions (continued)

Bits	Name	Description
22–23	HPIT	HPI priority position IPIC output interrupt type. Defines which type of IPIC output interrupt signal (*int*, *cint*, or *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 *int* request is asserted to the core for HPI. 01 *smi* request is asserted to the core for HPI. 10 *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).

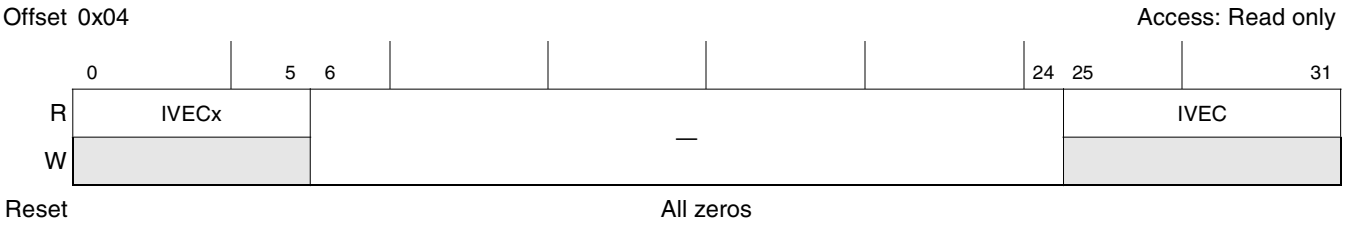


Figure 8-3. System Global Interrupt Vector Register (SIVCR)

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 SIVEC 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
1–8	Reserved	0b000_0001–0b000_1000
9	UART1	0b000_1001
10	UART2	0b000_1010
11	SEC	0b000_1011
12–13	Reserved	0b000_1100–0b000_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	IRQ4	0b001_0100
21	IRQ5	0b001_0101
22	IRQ6	0b001_0110
23	IRQ7	0b001_0111
24–31	Reserved	0b001_1000–0b001_1111
32	TSEC1 Tx	0b010_0000
33	TSEC1 Rx	0b010_0001
34	TSEC1 Err	0b010_0010
35	TSEC2 TX	0b010_0011
36	TSEC2 Rx	0b010_0100
37	TSEC2 Err	0b010_0101
38	USB DR	0b010_0110
39	USB MPH	0b010_0111
40–47	Reserved	0b010_1000–0b010_1111
48	IRQ0	0b011_0000
49–63	Reserved	0b011_0001–0b011_1111
64	RTC SEC	0b100_0000
65	PIT	0b100_0001
66	PCI1	0b100_0010
67	PCI2	0b100_0011
68	RTC ALR	0b100_0100
69	USB MPH	0b100_0101

Table 8-6. IVEC/CVEC/MVEC Field Definition (continued)

Interrupt ID Number	Interrupt Meaning	Interrupt Vector
70	SBA	0b100_0110
71	USB MPH	0b100_0111
72	GTM4	0b100_1000
73	GTM8	0b100_1001
74	GPIO1	0b100_1010
75	GPIO2	0b100_1011
76	DDR	0b100_1100
77	LBC	0b100_1101
78	GTM2	0b100_1110
79	GTM6	0b100_1111
80	PMC	0b101_0000
81–83	Reserved	0b101_0001–0b101_0011
84	GTM3	0b101_0100
85	GTM7	0b101_0101
86–89	Reserved	0b101_0110–0b101_1001
90	GTM1	0b101_1010
91	GTM5	0b101_1011
92–127	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, may be 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.

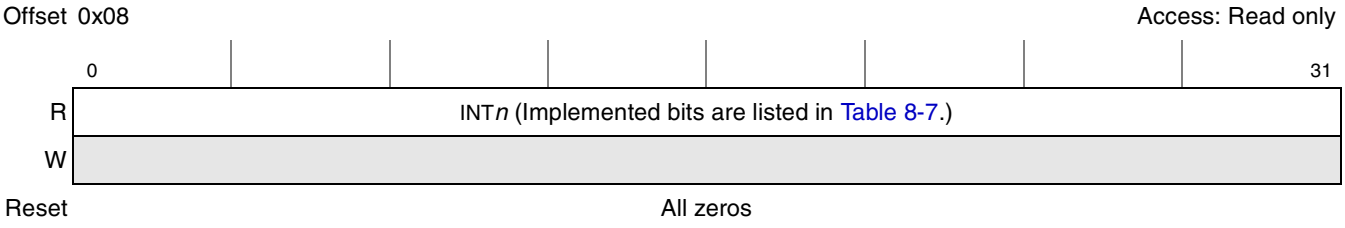


Figure 8-4. System Internal Interrupt Pending Register (SIPNR_H)

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	USB MPH
8–23	—
24	UART1
25	UART2
26	SEC
27–28	—
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*n*	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 8-4.

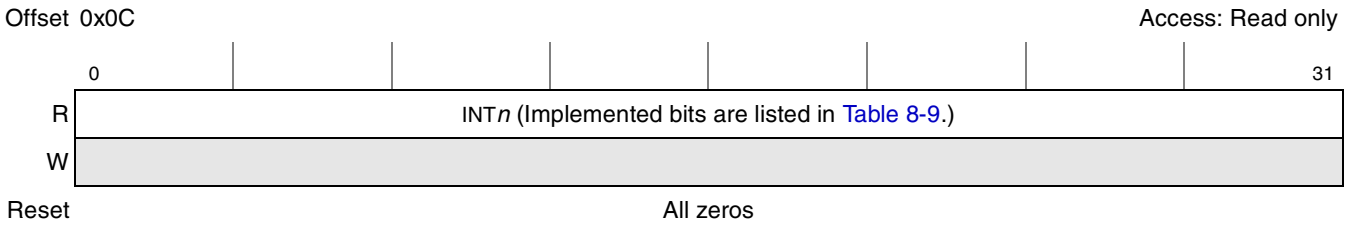


Figure 8-5. System Internal Interrupt Pending Register (SIPNR_L)

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	PCI1
3	PCI2
4	RTC ALR
5	MU
6	SBA
7	DMA
8	GTM4
9	GTM8
10	GPIO1
11	GPIO2
12	DDR
13	LBC
14	GTM2
15	GTM6
16	PMC
17–19	—
20	GTM3
21	GTM7
22–25	—
26	GTM1
27	GTM5
28–30	—
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), TSEC2 transmit request (TSEC2 Tx), TSEC2 receive request (TSEC2 Rx), TSEC2 transmit/receive error (TSEC2 Err), USB DR, and USB MPH internal interrupt signals.

For more information, see Section 8.6.3, “Internal Interrupts Group Relative Priority.”

Offset 0x10 Access: Read/write

	0–2	3–5	6–8	9–11	12–15	16–18	19–21	22–24	25–27	28–31
R W	SYSA0P	SYSA1P	SYSA2P	SYSA3P	—	SYSA4P	SYSA5P	SYSA6P	SYSA7P	—
Reset	0 0 0	0	1 0 1	0	0 1 1	0 0 0 0	1 0 0 1	0 1 1 1	0 1 1 1	0 0 0 0

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 USB MPH asserts its request in the SYSA0 position.
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 D Priority Register (SIPRR_D)

SIPRR_D, shown in Figure 8-7, defines the priority among the interrupt sources listed in Table 8-12.

Offset 0x1C Access: Read/write

	0–2	3–5	6–8	9–11	12–15	16–18	19–21	22–24	25–27	28–31
R W	SYSD0P	SYSD1P	SYSD2P	SYSD3P	—	SYSD4P	SYSD5P	SYSD6P	SYSD7P	—
Reset	0 0 0	0	1 0 1	0	0 1 1	0 0 0 0	1 0 0 1	0 1 1 1	0 1 1 1	0 0 0 0

Figure 8-7. System Internal Interrupt Group D Priority Register (SIPRR_D)

Table 8-12 defines the bit fields of SIPRR_D.

Table 8-12. 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 SEC asserts its request in the SYSD0 position. 011 Reserved 100 Reserved 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

8.5.6 System Internal Interrupt Mask Register (SIMSR_H and SIMSR_L)

Each implemented bit in SIMSR_H and SIMSR_L, shown in Figure 8-8 and Figure 8-9, 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 an SIMSR bit is cleared by the user at the same time 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.

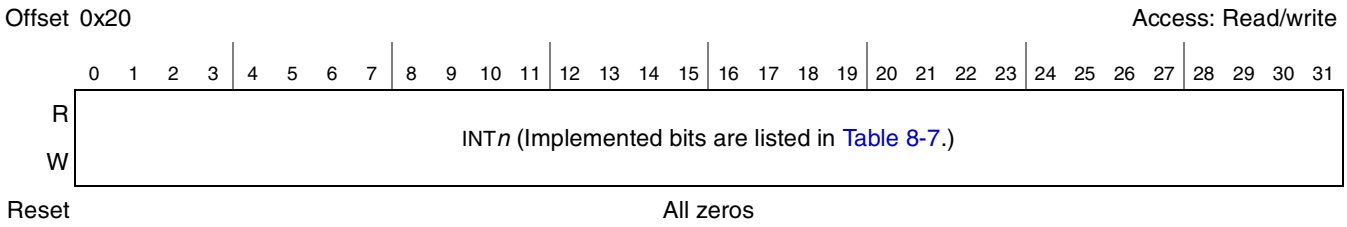


Figure 8-8. System Internal Interrupt Mask Register (SIMSR_H)

Table 8-13 defines the bit fields of SIMSR_H.

Table 8-13. SIMSR_H Field Descriptions

Bits	Name	Description
0–31	INT*n*	Each implemented bit (listed in Table 8-7) 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. **Note:** • SIMSR bit positions do not change according to their relative priority. • 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. • 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 **rfi** instruction. The error vector cannot be masked. Unimplemented bits, shown as reserved in Table 8-7, are ignored on writes; read = 0.

SIMSR_L is shown in Figure 8-9.

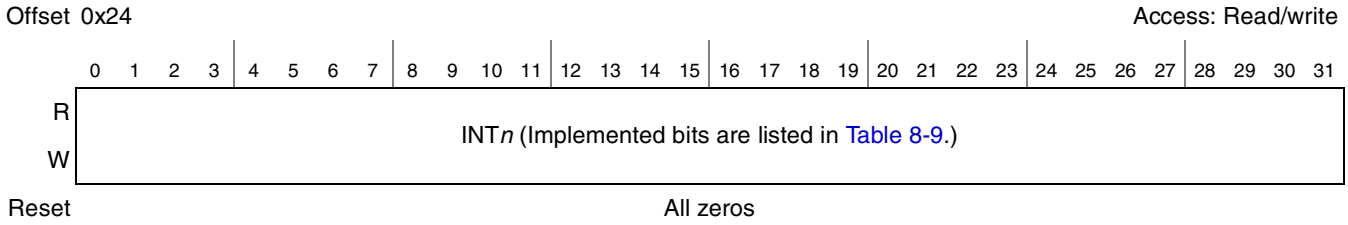


Figure 8-9. System Internal Interrupt Mask Register (SIMSR_L)

Table 8-14 defines the bit fields of SIMSR_L.

Table 8-14. SIMSR_L Field Descriptions

Bits	Name	Description
0–31	INT*n*	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. **Note:** • SIMSR bit positions are not changed according to their relative priority. • 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. • 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 Unimplemented bits, shown as reserved in Figure 8-9, are ignored on writes; read = 0.

8.5.7 System Internal Interrupt Control Register (SICNR)

SICNR, shown in Figure 8-10, defines the IPIC output interrupt type ($\overline{\text{int}}$, $\overline{\text{cint}}$, or $\overline{\text{smi}}$) in the SYSA0–SYSA1 and SYSD0–SYSD1 priority positions. All other priority positions assert $\overline{\text{int}}$ to the core.

Note that in core disabled mode the user should use the $\overline{\text{int}}$ output interrupt type (should not use $\overline{\text{cint}}$ or $\overline{\text{smi}}$ output interrupt types) to read an updated SIVCR.

Offset 0x28 Access: Read write

	0–1	2–3	4–23	24–25	26–27	28–31
R W	SYSD0T	SYSD1T	—	SYSA0T	SYSA1T	—
Reset	All zeros					

Figure 8-10. System Internal Interrupt Control Register (SICNR)

Table 8-15 defines the bit fields of SICNR.

Table 8-15. SICNR Field Descriptions

Bits	Name	Description
0–1	SYSD0T	SYSD0 priority position IPIC output interrupt type.Defines which type of the IPIC output interrupt signal (*int*, *cint*, or *smi*) 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). The definition of SYSD0T is as follows: 00 *int* request is asserted to the core for SYSD0. 01 *smi* request is asserted to the core for SYSD0. 10 *cint* request is asserted to the core for SYSD0. 11 Reserved
2–3	SYSD1T	Same as SYSD0T, but for SYSD1T.
4–23	—	Write ignored, read = 0
24–25	SYSA0T	SYSA0 priority position IPIC output interrupt type. Defines which type of the IPIC output interrupt signal (*int*, *smi*, or *cint*) asserts its request to the core in the SYSA0 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). The definition of SYSA0T is as follows: 00 *int* request is asserted to the core for SYSA0. 01 *smi* request is asserted to the core for SYSA0. 10 *cint* request is asserted to the core for SYSA0. 11 Reserved.
26–27	SYSA1T	Same as SYSA0T, but for SYSA1T
28–31	—	Write ignored, read = 0

8.5.8 System External Interrupt Pending Register (SEPNR)

Each bit in the SEPNR, shown in Figure 8-11, corresponds to an external interrupt source. When an interrupt is received, the interrupt controller sets the corresponding SEPNR bit.

Offset 0x2C Access: Read/write

	0	1	2	3	4	5	6	7	8–15
R	IRQ0[1]	IRQ1	IRQ2	IRQ3	IRQ4	IRQ5	IRQ6	IRQ7	—
W									
Reset	The reset values of implemented bits reflect the values of the external IRQ signals. Reserved bits are zeros. [2]								

	16–31
R	—
W	
Reset	All zeros

1 This bit is valid only if the IRQ0 signal is configured as an external maskable interrupt (SEMSR[SIRQ0] = 0)

2 The user should drive all IRQ inputs to an inactive state prior to reset negation

Figure 8-11. System External Interrupt Pending Register (SEPNR)

Table 8-16 defines the bit fields of SEPNR.

Table 8-16. SEPNR Field Descriptions

Bits	Name	Description
0–7	IRQ*n*	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 $\overline{\text{IRQn}}$ 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.
8–31	—	Write ignored, read = 0

8.5.9 System Mixed Interrupt Group A Priority Register (SMPRR_A)

The SMPRR_A, shown in Figure 8-12, defines the priority among the sources listed in Table 8-17.

Offset 0x30 Access: Read/write

	0–2	3–5	6–8	9–11	12–15	16–18	19–21	22–24	25–27	28–31
R	MIXA0P	MIXA1P	MIXA2P	MIXA3P	—	MIXA4P	MIXA5P	MIXA6P	MIXA7P	—
W										
Reset	0 0 0	0 1 0	1	0 0 1	1 0 0 0	1 0 0	1	0 1 1	1	0 1 1 1	0 0 0 0

Figure 8-12. System Mixed Interrupt Group A Priority Register (SMPRR_A)

Table 8-17 defines the bit fields of SMPRR_A.

Table 8-17. 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 PCI1 asserts its request to the MIXA0 position. 011 PCI2 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

8.5.10 System Mixed Interrupt Group B Priority Register (SMPRR_B)

SMPRR_B, shown in Figure 8-13, defines the priority among the sources listed in Table 8-18.

Offset 0x34 Access: Read/write

	0–2	3–5	6–8	9–11	12–15	16–18	19–21	22–24	25–27	28–31
R W	MIXB0P	MIXB1P	MIXB2P	MIXB3P	—	MIXB4P	MIXB5P	MIXB6P	MIXB7P	—
Reset	0 0 0 0	0 1 0 1	0 0 1 1	0 0 0 0	1 0 0 1	0 1 1 1	0 1 1 1	0 0 0 0		

Figure 8-13. System Mixed Interrupt Group B Priority Register (SMPRR_B)

Table 8-18 defines the bit fields of SMPRR_B.

Table 8-18. SMPRR_B Field Descriptions

Bits	Name	Description
0–2 3–11, 16–27	MIXB*n*P	MIXB*n* priority order. Defines which interrupt source asserts its request in the MIXB*n* 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 MIXB*n*P is as follows: 000 RTC ALR asserts its request to the MIXB*n* position. 001 MU asserts its request to the MIXB*n* position. 010 SBA asserts its request to the MIXB*n* position. 011 DMA asserts its request to the MIXB*n* position. 100 IRQ4 asserts its request to the MIXB*n* position. 101 IRQ5 asserts its request to the MIXB*n* position. 110 IRQ6 asserts its request to the MIXB*n* position. 111 IRQ7 asserts its request to the MIXB*n* position.
12–15, 28–31	—	Write ignored, read = 0

8.5.11 System External Interrupt Mask Register (SEMSR)

Each bit in the system external interrupt mask register (SEMSR), shown in Figure 8-11, 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 SEPNR 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 an SEMSR bit is cleared by the user at the same time that 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

	0	1	2	3	4	5	6	7	8 – 15
R W	IRQ0[1]	IRQ1	IRQ2	IRQ3	IRQ4	IRQ5	IRQ6	IRQ7	—
Reset	The reset values of implemented bits reflect the values of the external IRQ signals. Reserved bits are zeros. [2]								

	16	17 – 31
R W	SIRQ0	—
Reset	All zeros	

1 This bit is valid only if the IRQ0 signal is configured as an external maskable interrupt (SEMSR[SIRQ0] = 0)

2 The user should drive all IRQ inputs to an inactive state prior to reset negation

Figure 8-14. System External Interrupt Mask Register (SEMSR)

Table 8-19 defines the bit fields of SEMSR.

Table 8-19. SEMSR Field Descriptions

Bits	Name	Description
0–7	—	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. **Note:** • SEMSR bit positions are not affected by their relative priority. • 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. • If an SEMSR bit is masked at the same time that the corresponding SEPNR 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.
8–15	—	Write ignored, read = 0
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

8.5.12 System External Interrupt Control Register (SECNR)

SECNR, shown in Figure 8-15, defines the edge detect mode for external $\overline{\text{IRQ}}n$ interrupt signals and determines whether the corresponding $\overline{\text{IRQ}}n$ 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 ($\overline{\text{int}}$, $\overline{\text{cint}}$, or $\overline{\text{smi}}$) 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

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MIXB0T		MIXB1T		—				MIXA0T		MIXA1T		—			
Reset	All zeros															

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	EDI0	EDI1	EDI2	EDI3	EDI4	EDI5	EDI6	EDI7	—							
Reset	All zeros															

Figure 8-15. System External Interrupt Control Register (SECNR)

Table 8-20 defines the bit fields of SECNR.

Table 8-20. 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 (*int*, *cint*, or *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 *int* request is asserted to the core for MIXB0. 01 *smi* request is asserted to the core for MIXB0. 10 *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
8–9	MIXA0T	MIXA0 priority position IPIC output interrupt Type. Defines which type of the IPIC output interrupt signal (*int*, *cint*, or *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 *int* request is asserted to the core for MIXA0. 01 *smi* request is asserted to the core for MIXA0. 10 *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
16–23	EDIx	Each bit defines the edge detect mode for the external $\overline{\text{IRQ}}n$ interrupt signals, determines whether the corresponding $\overline{\text{IRQ}}n$ signal asserts an interrupt request upon either a high-to-low change or low assertion on the pin. The corresponding $\overline{\text{IRQ}}n$ signal asserts an interrupt request as follows: 0 Low assertion on $\overline{\text{IRQ}}n$ generates an interrupt request (level sensitive). 1 High-to-low change on $\overline{\text{IRQ}}n$ generates an interrupt request (edge sensitive).
24–31	—	Write ignored, read = 0

8.5.13 System Error Status Register (SERSR)

Each implemented bit in SERSR, shown in Figure 8-16, corresponds to an external and an internal non-maskable error source machine check (mcp). When an error interrupt signal is received, the interrupt controller sets the corresponding SERSR bit.

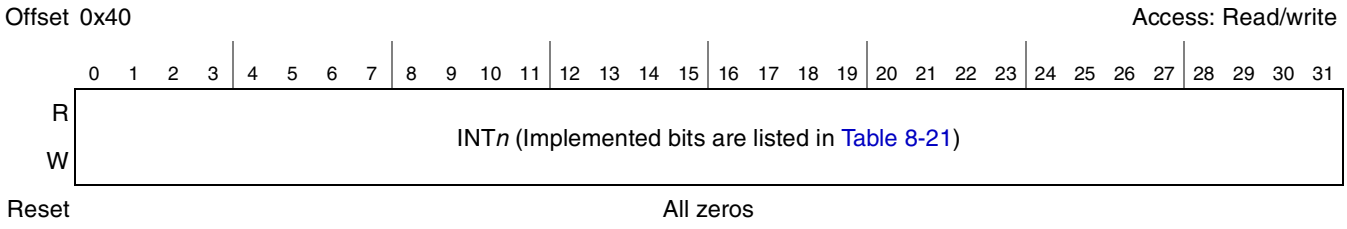


Figure 8-16. System Error Status Register (SERSR)

Table 8-21 lists the implemented SERSR bits. Note that these field assignments are valid for SERMR and SERFR as well.

Table 8-21. SERSR/SERMR/SERFR Bit Assignments

Bits	Field
0	IRQ0[1]
1	WDT
2	SBA
3	—
4	—
5	PCI1
6	PCI2
7	MU
8–14	—
15	—
16–31	—

[1] This bit is valid only if the IRQ0 signal is configured as an external MCP interrupt (SEMSR[SIRQ0] = 1)

Table 8-22 defines the bit fields of SERSR.

Table 8-22. SERSR Field Descriptions

Bits	Name	Description
0–31	INT*n*	Each implemented bit in the SERSR, listed in Table 8-21, 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 8-21), writes are ignored, read = 0

8.5.14 System Error Mask Register (SERMR)

Each implemented bit in SERMR, shown in Figure 8-17, 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.

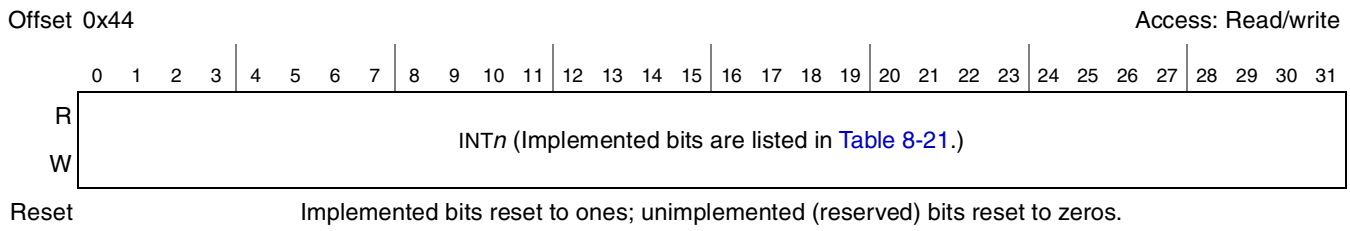


Figure 8-17. System Error Mask Register (SERMR)

Table 8-23 defines the bit fields of SERMR.

Table 8-23. SERMR Field Descriptions

Bits	Name	Description
0–31	INT*n*	Each implemented SERMR bit, listed in Table 8-21, 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

8.5.15 System Error Control Register (SERCR)

SERCR, shown in Figure 8-18, defines the control bits that route MCP requests in core disable mode to either $\overline{\text{MCP_OUT}}$ or $\overline{\text{INTA}}$ in core-disable mode.

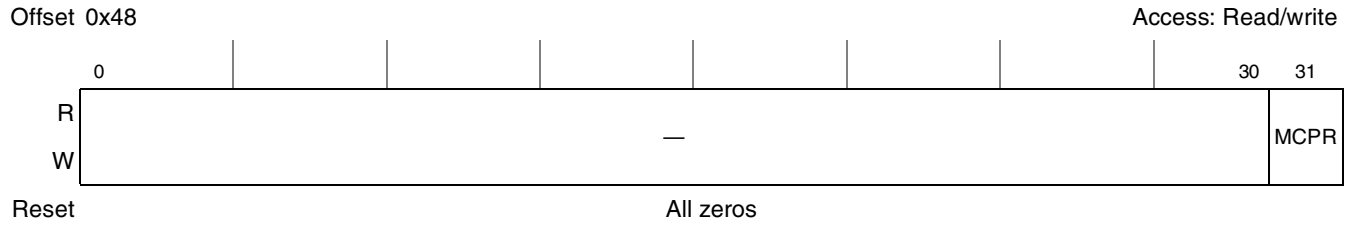


Figure 8-18. System Error Control Register (SERCR)

Table 8-24 defines the bit fields of SERCR.

Table 8-24. SERCR Field Descriptions

Bits	Name	Description
0–30	—	Write ignored, read = 0
31	MCPR	MCP route. Route MCP request to either $\overline{\text{MCP_OUT}}$ or $\overline{\text{INTA}}$ (in core disable mode). 0 MCP routed to $\overline{\text{INTA}}$ (in core disable mode). 1 MCP routed to $\overline{\text{MCP_OUT}}$ (in core disable mode).

8.5.16 System Internal Interrupt Force Registers (SIFCR_H and SIFCR_L)

Each implemented bit SIFCR_H and SIFCR_L, shown in Figure 8-19 and Figure 8-20, 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.

Figure 8-19. System Internal Interrupt Force Register (SIFCR_H)

Table 8-25 defines the bit fields of SIFCR_H.

Table 8-25. SIFCR_H Field Descriptions

Bits	Name	Description
0–31	INT*n*	Each implemented bit, listed in Table 8-7, corresponds to an external interrupt source. The user forces an interrupt by setting the corresponding SIFCR*x* bit. SIFCR*n* 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 8-20.

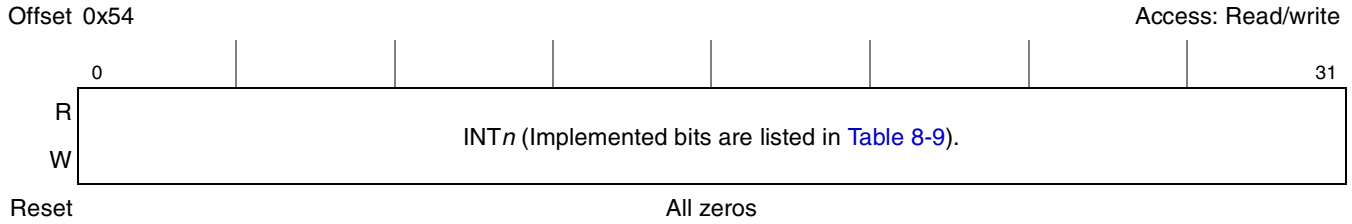


Figure 8-20. System Internal Interrupt Force Register (SIFCR_L)

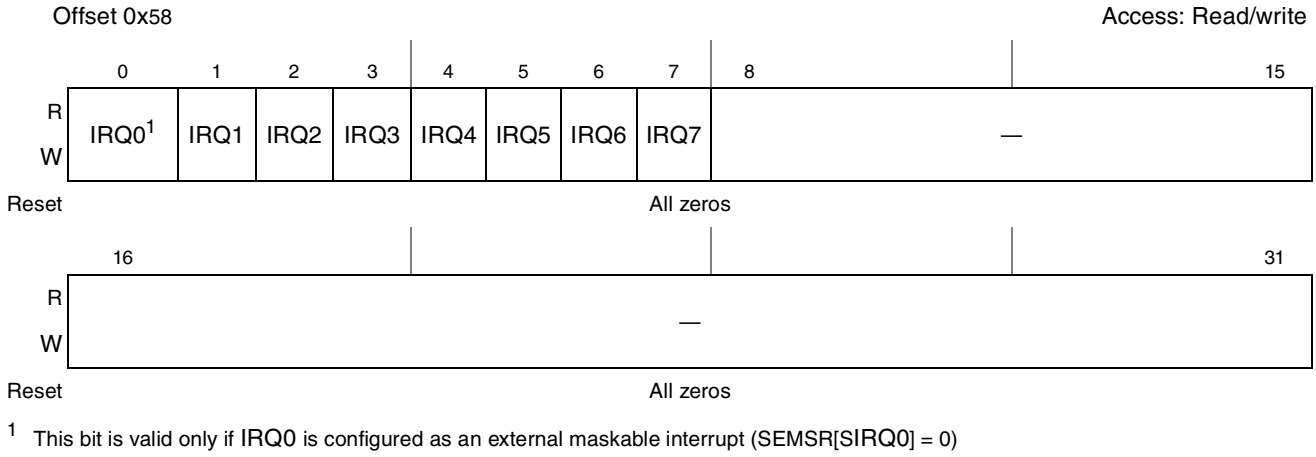
Table 8-26 defines the bit fields of SIFCR_L.

Table 8-26. SIFCR_L Field Descriptions

Bits	Name	Description
0–31	INT*n*	Each implemented bit, listed in Table 8-9, corresponds to an external 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

8.5.17 System External Interrupt Force Register (SEFCR)

Each implemented bit in SEFCR, shown in Figure 8-21, corresponds to an external interrupt source. When a bit is set, the interrupt controller generates the corresponding external interrupt (sets the corresponding SEPNR bit). SEFCR can be read by the user at any time.



[1] This bit is valid only if IRQ0 is configured as an external maskable interrupt (SEMSR[SIRQ0] = 0)

Figure 8-21. System External Interrupt Force Register (SEFCR)

Table 8-27 defines the bit fields of SEFCR.

Table 8-27. SEFCR Field Descriptions

Bits	Name	Description
0–7	IRQ*n*	Each bit corresponds to an external interrupt source. The user force an interrupt by setting the SEFCR bit. **Note:** SEFCR bit positions are not affected by their relative priority.
8–31	—	Write ignored, read = 0

8.5.18 System Error Force Register (SERFR)

Each bit in the system error force register (SERFR), shown in Figure 8-22, 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 8-22. System Error Status Register (SERFR)

Table 8-28 defines the bit fields of SERFR.

Table 8-28. SERFR Field Descriptions

Bits	Name	Description
0–31	INT*n*	Each implemented bit, listed in Table 8-21, 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

8.5.19 System Critical Interrupt Vector Register (SCVCR)

SCVCR, shown in Figure 8-23, contains a 7-bit code (Table 8-29) representing the unmasked critical interrupt ($\overline{\text{CINT}}$) 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).

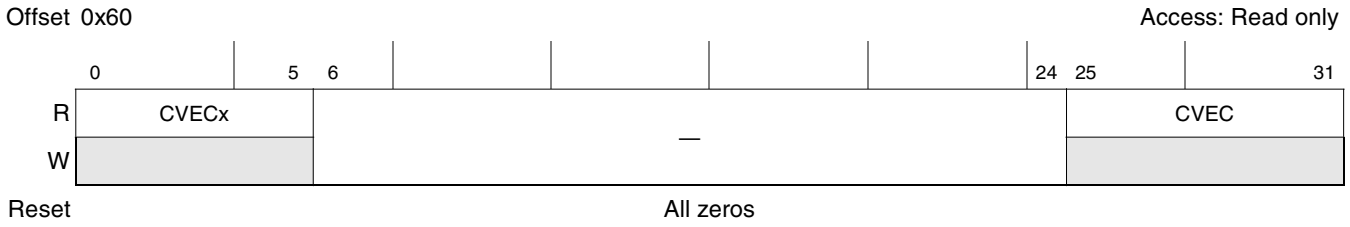


Figure 8-23. System Critical Interrupt Vector Register (SCVCR)

Table 8-29 defines SCVCR bit fields.

Table 8-29. 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.

8.5.20 System Management Interrupt Vector Register (SMVCR)

SMVCR, shown in Figure 8-24, contains a 7-bit code (Table 8-30) representing the unmasked system management interrupt ($\overline{\text{SMI}}$) 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).

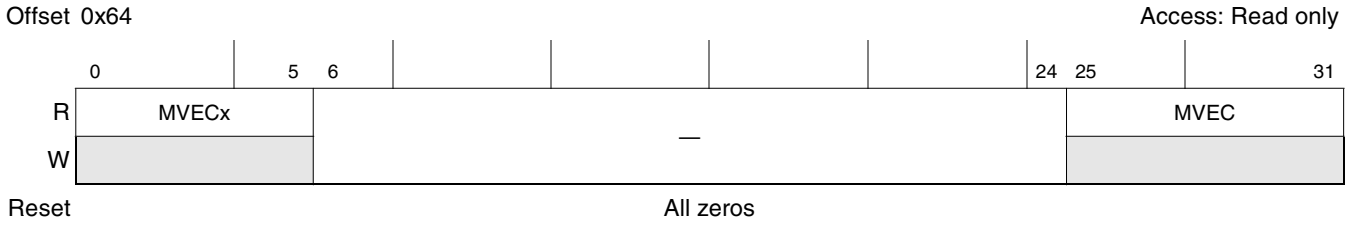


Figure 8-24. System Management Interrupt Vector Register (SMVCR)

Table 8-30 defines the bit fields of SMVCR.

Table 8-30. 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 8-25 shows the interrupt configuration.

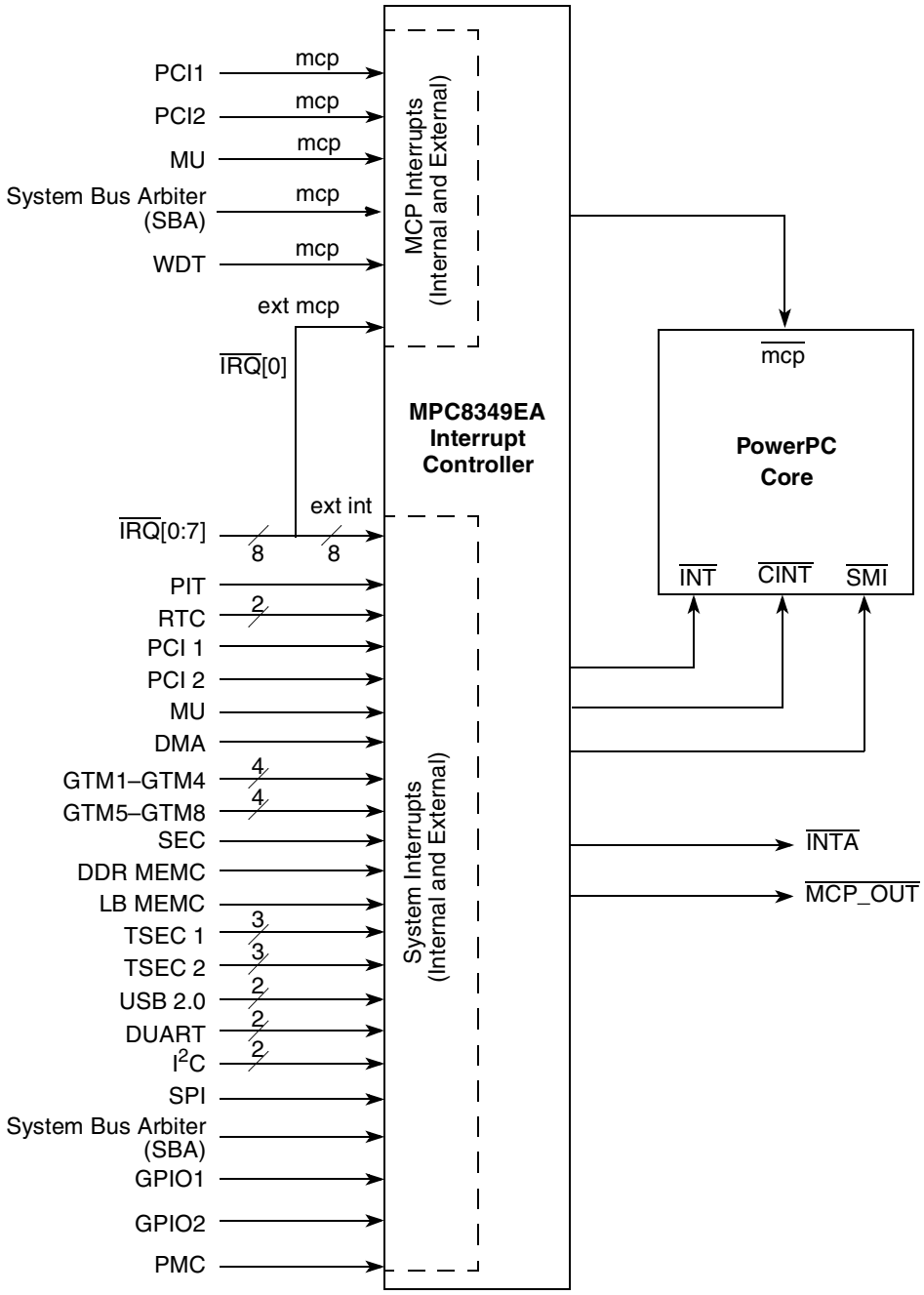


Figure 8-25. Interrupt Structure

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 PowerPC 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 TSEC1 Tx, TSEC1 Rx, TSEC1 Err, TSEC2 TX, TSEC2 Rx USB DR, and USB MPH internal interrupt signals can be modified.
- The relative priority of the UART1, UART2, SPI, I2C1, I2C2, and SEC internal interrupt signals can be modified.
- The relative priority of the IRQ0, IRQ1, IRQ2, and IRQ3 external interrupts, and RTC SECPCI1, and PCI2 internal interrupts can be modified.
- The relative priority of the IRQ4, IRQ5, IRQ6, and IRQ7 external interrupts, and RTC ALR, MU, SBA, and DMA 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*x*) 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 8-31, 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 8-31 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*x*) 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. See Table 8-31 for more information. 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 8-31.

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 8-31. 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 8-31 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 8-31. 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)
7–10	Reserved
11	MIXA1 (Spread)
12–15	Reserved
16	MIXB0 (Grouped)
17	MIXB1 (Grouped)
18	MIXB2 (Grouped)
19	MIXB3 (Grouped)
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)

Table 8-31. Interrupt Source Priority Levels (continued)

Priority	Interrupt Source Description
30	MIXA4 (Grouped)
31	MIXA5 (Grouped)
32	MIXA6 (Grouped)
33	MIXA7 (Grouped)
34	MIXB2 (Spread)
35–38	Reserved
39	MIXA3 (Spread)
40–43	Reserved
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)
53	MIXA4 (Spread)
54	SYSD4 (Grouped)
55	SYSD5 (Grouped)
56	SYSD6 (Grouped)
57	SYSD7 (Grouped)
58	MIXB4 (Spread)
59	GTM4
60	Reserved
61	SYSA0 (Spread)
62	GTM8
63	Reserved
64	SYSD0 (Spread)
65	Reserved
66	GPIO1
67	MIXA5 (Spread)
68	GPIO2
69	Reserved
70	SYSA1 (Spread)

Table 8-31. Interrupt Source Priority Levels (continued)

Priority	Interrupt Source Description
71	DDR
72	Reserved
73	SYSD1 (Spread)
74	Reserved
75	LBC
76	MIXB5 (Spread)
77	GTM2
78	Reserved
79	SYSA2 (Spread)
80	GTM6
81	Reserved
82	SYSD2 (Spread)
83	Reserved
84	PMC
85	MIXA6 (Spread)
86	Reserved
87	Reserved
88	SYSA3 (Spread)
89	Reserved
90	Reserved
91	SYSD3 (Spread)
92	Reserved
93	Reserved
94	MIXB6 (Spread)
95	GTM3
96	Reserved
97	SYSA4 (Spread)
98	GTM7
99	Reserved
100	SYSD4 (Spread)
101	Reserved
102	Reserved
103	MIXA7 (Spread)
104	Reserved
105	Reserved
106	SYSA5 (Spread)

Table 8-31. Interrupt Source Priority Levels (continued)

Priority	Interrupt Source Description
107	Reserved
108	Reserved
109	SYSD5 (Spread)
110	Reserved
111	Reserved
112	MIXB7 (Spread)
113	GTM1
114	Reserved
115	SYSA6 (Spread)
116	GTM5
117	Reserved
118	SYSD6 (Spread)
119	Reserved
120	Reserved
121	Reserved
122	Reserved
123	SYSA7 (Spread)
124	Reserved
125	Reserved
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 8-31 shows which interrupt sources have multiple interrupting events.

Figure 8-26 shows an example of how the masking occurs, using a DDR block as an example.

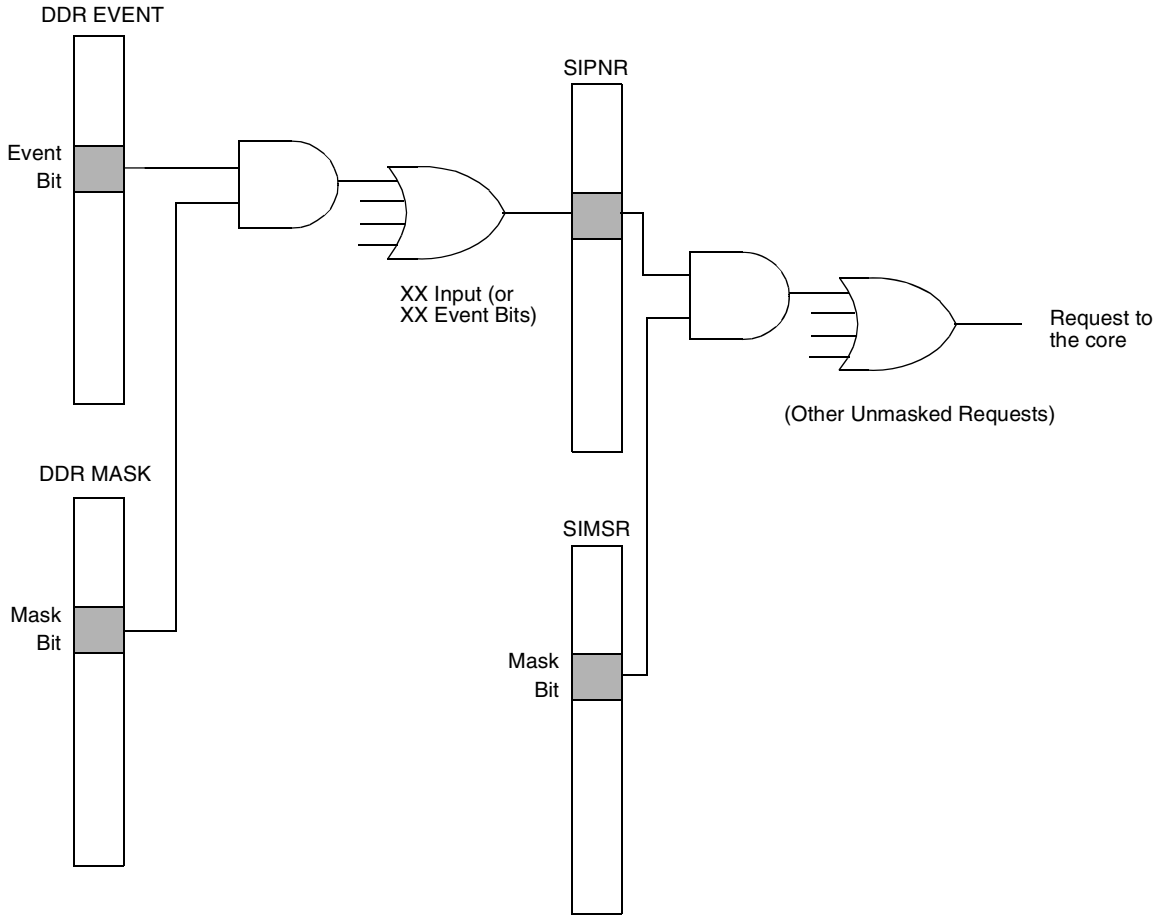


Figure 8-26. DDR Interrupt Request Masking

8.6.8 Interrupt Vector Generation and Calculation

Pending unmasked interrupts are presented to the core in order of priority according to Table 8-31. 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 PIC 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 8-21.

Chapter 9
DDR Memory Controller

9.1 Introduction

The fully programmable DDR SDRAM controller supports most JEDEC standard x8 or x16 DDR and 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 two or three bits on the bank address (MBA) pins during a memory access.

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.

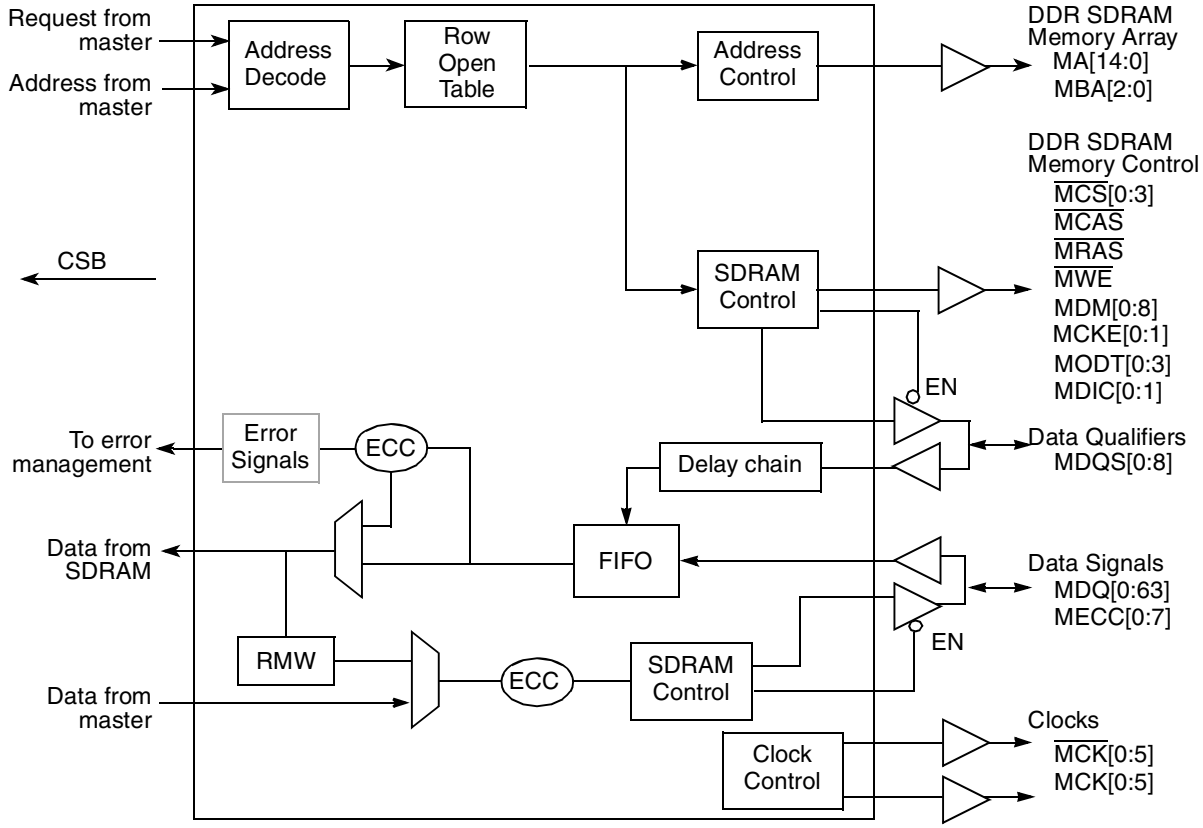


Figure 9-1. DDR Memory Controller Simplified Block Diagram

9.2 Features

The DDR memory controller includes these distinctive features:

- Support for DDR2 and DDR SDRAM
- 64/72-bit SDRAM data bus. 32/40-bit SDRAM data bus supported
- Programmable settings for meeting all SDRAM timing parameters
- The following SDRAM configurations are supported:
  - As many as four physical banks (chip selects), each bank independently addressable
  - 64-Mbit to 4-Gbit devices depending on internal device configuration with x8/x16/x32 data ports (no direct x4 support)
  - Unbuffered and registered DIMMs
- Chip select interleaving support
- 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*n*_CONFIG[AP_*n*_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.

NOTE

A bar over a signal name indicates that the signal is active low, such as $\overline{\text{MCAS}}$ (column address strobe). 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 MDQ (data bus), are referred to as asserted when they are high and negated when they are low.

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[0:63]	Data bus	All zeros	64	I/O
MDQS[0:8]	Data strobes	All zeros	9	I/O
MECC[0:7]	Error checking and correcting	All zeros	8	I/O
$\overline{\text{MCAS}}$	Column address strobe	One	1	O
MA[14:0]	Address bus	All zeros	15	O
MBA[2:0]	Logical bank address	All zeros	3	O
$\overline{\text{MCS}}$[0:3]	Chip selects	All ones	4	O
$\overline{\text{MWE}}$	Write enable	One	1	O
$\overline{\text{MRAS}}$	Row address strobe	One	1	O
MDM[0:8]	Data mask	All zeros	9	O
MCK[0:5]	DRAM clock outputs	All zeros	6	O
$\overline{\text{MCK}}$[0:5]	DRAM clock outputs (complement)	All zeros	6	O
MCKE[0:1]	DRAM clock enable	All zeros	2	O
MODT[0:3]	DRAM on-die termination	All zeros	4	O
MDVAL	Memory debug data valid	Zero	1	O
MSRCID[0:4]	Memory debug source ID	All zeros	5	O
MDIC[0:1]	Driver impedance calibration	High Z	2	I/O

Table 9-2 shows the memory address signal mappings.

Table 9-2. Memory Address Signal Mappings

Signal Name (Outputs)		JEDEC DDR DIMM Signals (Inputs)		Signal Name (Outputs)		JEDEC DDR DIMM Signals (Inputs)
msb	MA14	A14			MA4	A4
	MA13	A13			MA3	A3
	MA12	A12			MA2	A2
	MA11	A11			MA1	A1
	MA10	A10 (AP for DDR)		lsb	MA0	A0
	MA9	A9				
	MA8	A8				
	MA7	A7		msb	MBA2	MBA2
	MA6	A6			MBA1	MBA1
	MA5	A5		lsb	MBA0	MBA0

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[0:63]	I/O	Data bus. Both input and output signals on the DDR memory controller.	
	O	As outputs for the bidirectional data bus, these signals operate as described below.	
		State Meaning	Asserted/Negated—Represent the value of data being driven by the DDR memory controller.
		Timing	Assertion/Negation—Driven coincident with corresponding data strobes (MDQS) signal. 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 a s described below.	
		State Meaning	Asserted/Negated—Represents the state of data being driven by the external DDR SDRAMs.
		Timing	Assertion/Negation—The DDR SDRAM drives data during a READ transaction. High-impedance—No READ or WRITE command in progress; data is not being driven by the memory controller or the DRAM.
MDQS[0:8]/	I/O	Data strobes. Inputs with read data and as outputs with write data. The data strobes may be single ended or differential.	
	O	As outputs, the data strobes are driven by the DDR memory controller during a write transaction. The memory controller always drives these signals low unless a read has been issued and incoming data strobes are expected. This keeps the data strobes from floating high when there are no transactions on the DRAM interface.	
		State Meaning	Asserted/Negated—Driven high when positive capture data is transmitted and driven low when negative capture data is transmitted. Centered in the data “eye” for writes; coincident with the data eye for reads. Treated as a clock. Data is valid when signals toggle. See Table 9-36 for byte lane assignments.
		Timing	Assertion/Negation—If a WRITE command is registered at clock edge *n*, data strobes at the DRAM assert centered in the data eye on clock edge *n* + 1. See the JEDEC DDR SDRAM specification for more information.
	I	As inputs, the data strobes are driven by the external DDR SDRAMs during a read transaction. The data strobes are used by the memory controller to synchronize data latching.	
		State Meaning	Asserted/Negated—Driven high when positive capture data is received and driven low when negative capture data is received. Centered in the data eye for writes; coincident with the data eye for reads. Treated as a clock. Data is valid when signals toggle. See Table 9-36 for byte lane assignments.
		Timing	Assertion/Negation—If a READ command is registered at clock edge *n*, and the latency is programmed in TIMING_CFG_1[CASLAT] to be *m* clocks, data strobes at the DRAM assert coincident with the data on clock edge *n* + *m*. See the JEDEC DDR SDRAM specification for more information.

Table 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	As normal mode outputs the ECC signals represent the state of ECC driven by the DDR controller on writes. As debug mode outputs MECC[0:5] provide source ID and data-valid information. See Section 9.5.11, "Error Checking and Correcting (ECC)," and Section 5.3.2.7, "Debug Configuration," for more details.	
		State Meaning	Asserted/Negated—Represents the state of ECC being driven by the DDR controller on writes.
		Timing	Assertion/Negation—Same timing as MDQ High-impedance—Same timing as MDQ
	I	As inputs, the ECC signals represent the state of ECC driven by the SDRAM devices on reads.	
		State Meaning	Asserted/Negated—Represents the state of ECC being driven by the DDR SDRAMs on reads.
		Timing	Assertion/Negation—Same timing as MDQ High-impedance—Same timing as MDQ
MA[14:0]	O	Address bus. Memory controller outputs for the address to the DRAM. MA[14:0] carry 15 of the address bits for the DDR memory interface corresponding to the row and column address bits. MA0 is the lsb of the address output from the memory controller.	
		State Meaning	Asserted/Negated—Represents the address driven by the DDR memory controller. Contains different portions of the address depending on the memory size and the DRAM command being issued by the memory controller. See Table 9-39 and Table 9-41 for a complete description of the mapping of these signals.
		Timing	Assertion/Negation—The address is always driven when the memory controller is enabled. It is valid when a transaction is driven to DRAM (when $\overline{\text{MCS}}n$ 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.	
		State Meaning	Asserted/Negated—Selects the DDR SDRAM logical (or internal) bank to be activated during the row address phase and selects the SDRAM internal bank for the read or write operation during the column address phase of the memory access. Table 9-39 and Table 9-41 describe the mapping of these signals in all cases.
		Timing	Assertion/Negation—Same timing as MA*n* High-impedance—Same timing as MA*n*

Table 9-3. Memory Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
$\overline{\text{MCAS}}$	O	Column address strobe. Active-low SDRAM address multiplexing signal. $\overline{\text{MCAS}}$ is asserted for read or write transactions and for mode register set, refresh, and precharge commands.	
		State Meaning	Asserted—Indicates that a valid SDRAM column address is on the address bus for read and write transactions. See Table 9-45 for more information on the states required on $\overline{\text{MCAS}}$ for various other SDRAM commands. Negated—The column address is not guaranteed to be valid.
		Timing	Assertion/Negation—Assertion and negation timing is directed by the values described in Section 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)." High-impedance—$\overline{\text{MCAS}}$ is always driven unless the memory controller is disabled.
$\overline{\text{MRAS}}$	O	Row address strobe. Active-low SDRAM address multiplexing signal. Asserted for activate commands. In addition; used for mode register set commands and refresh commands.	
		State Meaning	Asserted—Indicates that a valid SDRAM row address is on the address bus for read and write transactions. See Table 9-45 for more information on the states required on $\overline{\text{MRAS}}$ for various other SDRAM commands. Negated—The row address is not guaranteed to be valid.
		Timing	Assertion/Negation—Timing is directed by the values 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)." High-impedance—$\overline{\text{MRAS}}$ is always driven unless the memory controller is disabled.
$\overline{\text{MCS}}$[0:3]	O	Chip selects. Four chip selects supported by the memory controller.	
		State Meaning	Asserted—Selects a physical SDRAM bank to perform a memory operation as described in Section 9.4.1.1, "Chip Select Memory Bounds (CSn_BNDS)," and Section 9.4.1.2, "Chip Select Configuration (CSn_CONFIG)." The DDR controller asserts one of the $\overline{\text{MCS}}$[0:3] signals to begin a memory cycle. Negated—Indicates no SDRAM action during the current cycle.
		Timing	Assertion/Negation—Asserted to signal any new transaction to the SDRAM. The transaction must adhere to the timing constraints set in TIMING_CFG_0–TIMING_CFG_3. High-impedance—Always driven unless the memory controller is disabled.
$\overline{\text{MWE}}$	O	Write enable. Asserted when a write transaction is issued to the SDRAM. This is also used for mode registers set commands and precharge commands.	
		State Meaning	Asserted—Indicates a memory write operation. See Table 9-45 for more information on the states required on $\overline{\text{MWE}}$ for various other SDRAM commands. Negated—Indicates a memory read operation.
		Timing	Assertion/Negation—Similar timing as $\overline{\text{MRAS}}$ and $\overline{\text{MCAS}}$. Used for write commands. High-impedance—$\overline{\text{MWE}}$ is always driven unless the memory controller is disabled.

Table 9-3. Memory Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description
MDM[0:8]	O	DDR SDRAM data output mask. Masks unwanted bytes of data transferred during a write. They are needed to support sub-burst-size transactions (such as single-byte writes) on SDRAM where all I/O occurs in multi-byte bursts. MDM0 corresponds to the most significant byte (MSB) and MDM7 corresponds to the LSB, while MDM8 corresponds to the ECC byte. Table 9-36 shows byte lane encodings.
	State Meaning	Asserted—Prevents writing to DDR SDRAM. Asserted when data is written to DRAM if the corresponding byte(s) should be masked for the write. Note that the MDM*n* signals are active-high for the DDR controller. MDM*n* is part of the DDR command encoding. Negated—Allows the corresponding byte to be read from or written to the SDRAM.
	Timing	Assertion/Negation—Same timing as MDQx as outputs. High-impedance—Always driven unless the memory controller is disabled.
MODT[0:3]	O	On-Die termination. Memory controller outputs for the ODT to the DRAM. MODT[0:3] represents the on-die termination for the associated data, data masks, ECC, and data strobes.
	State Meaning	Asserted/Negated—Represents the ODT driven by the DDR memory controller.
	Timing	Assertion/Negation—Driven in accordance with JEDEC DRAM specifications for on-die termination timings. It is configured through the CS*n*_CONFIG[ODT_RD_CFG] and CS*n*_CONFIG[ODT_WR_CFG] fields. High-impedance—Always driven.
MDIC[0:1]	I/O	Driver impedance calibration. Note that the MDIC signals require the use of 18.2-Ω precision 1% resistors; MDIC0 must be pulled to GND, while MDIC1 must be pulled to GV_{DD}. See Section 5.3.2.8, "DDR Control Driver Register (DDRCDR)," for more information on these signals.
	State Meaning	These pins are used for automatic calibration of the DDR IOs.
	Timing	These will be driven for four DRAM cycles at a time while the DDR controller is executing the automatic driver compensation.

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[0:5], $\overline{\text{MCK}}$[0:5]	O	DRAM clock outputs and their complements. See Section 9.5.4.1, "Clock Distribution." Enabled by the MCKENR register. See Section 4.5.3.1, "MCK Enable Register (MCKENR)," for details.
	State Meaning	Asserted/Negated—The JEDEC DDR SDRAM specifications require true and complement clocks. A clock edge is seen by the SDRAM when the true and complement cross.
	Timing	Assertion/Negation—Timing is controlled by the DDR_CLK_CNTL register at offset 0x130.
MCKE[0:1]	O	Clock enable. Output signals used as the clock enables to the SDRAM. MCKE[0:1] can be negated to stop clocking the DDR SDRAM.
	State Meaning	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[0:1] are negated.
	Timing	Assertion/Negation—Asserted when DDR_SDRAM_CFG[MEM_EN] is set. Can be negated when entering dynamic power management or self refresh. Will be asserted again when exiting dynamic power management or self refresh. High-impedance—Always driven.

9.3.2.3 Debug Signals

The debug signals MSRCID[0:4] and MDVAL have no function in normal DDR controller operation. A detailed description of these signals can be found in: Section 5.3.2.7, “Debug Configuration.”

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
DDR Registers—Block Base Address: 0x0_2000				
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
0x010	CS2_BNDS—Chip select 2 memory bounds	R/W	0x0000_0000	9.4.1.1/9-10
0x018	CS3_BNDS—Chip select 3 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
0x088	CS2_CONFIG—Chip select 2 configuration	R/W	0x0000_0000	9.4.1.2/9-11
0x08C	CS3_CONFIG—Chip select 3 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	9.4.1.7/9-20
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	9.4.1.9/9-24
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

Table 9-5. DDR Memory Controller Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x140–0x147	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	0x*nnnn_nnnn*[1]	9.4.1.16/9-30
0xBFC	DDR_IP_REV2—DDR IP block revision 2	R	0x0000_0000	9.4.1.17/9-30
0xE00	DATA_ERR_INJECT_HI—Memory data path error injection mask high	R/W	0x0000_0000	9.4.1.18/9-31
0xE04	DATA_ERR_INJECT_LO—Memory data path error injection mask low	R/W	0x0000_0000	9.4.1.19/9-31
0xE08	ECC_ERR_INJECT—Memory data path error injection mask ECC	R/W	0x0000_0000	9.4.1.20/9-32
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	9.4.1.22/9-33
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
0xE58	ERR_SBE—Single-Bit ECC memory error management	R/W	0x0000_0000	9.4.1.29/9-37

[1] 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*n*_BNDS)

The chip select bounds registers (CS*n*_BNDS) define the starting and ending address of the memory space that corresponds to the individual chip selects. Note that the size specified in CS*n*_BNDS should equal the size of physical DRAM. Also, note that EA*n* must be greater than or equal to SA*n*.

If chip select interleaving is enabled, all fields in the lower interleaved chip select will be used, and the other chip selects' bounds registers will be unused. For example, if chip selects 0 and 1 are interleaved, all fields in CS0_BNDS will be used, and all fields in CS1_BNDS will be unused.

CS*n*_BNDS are shown in Figure 9-2.

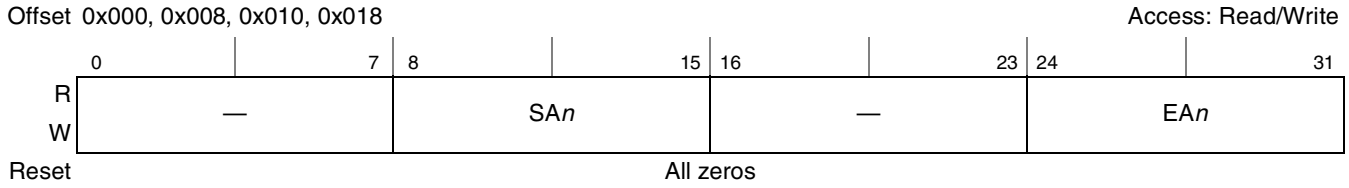


Figure 9-2. Chip Select Bounds Registers (CS*n*_BNDS)

Table 9-6 describes the CS*n*_BNDS register fields.

Table 9-6. CS*n*_BNDS Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8–15	SA*n*	Starting address for chip select (bank) *n*. This value is compared against the 8 msbs of the 32-bit address.
16–23	—	Reserved
24–31	EA*n*	Ending address for chip select (bank) *n*. This value is compared against the 8 msbs of the 32-bit address.

9.4.1.2 Chip Select Configuration (CS*n*_CONFIG)

The chip select configuration (CS*n*_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*n*_CONFIG[ROW_BITS_CS_*n*, COL_BITS_CS_*n*] establish address multiplexing, the user should take great care to set these values correctly.

If chip select interleaving is enabled, then all fields in the lower interleaved chip select will be used, and the other registers' fields will be 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 will be used, but only the ODT_RD_CFG and ODT_WR_CFG fields in CS1_CONFIG will be used.

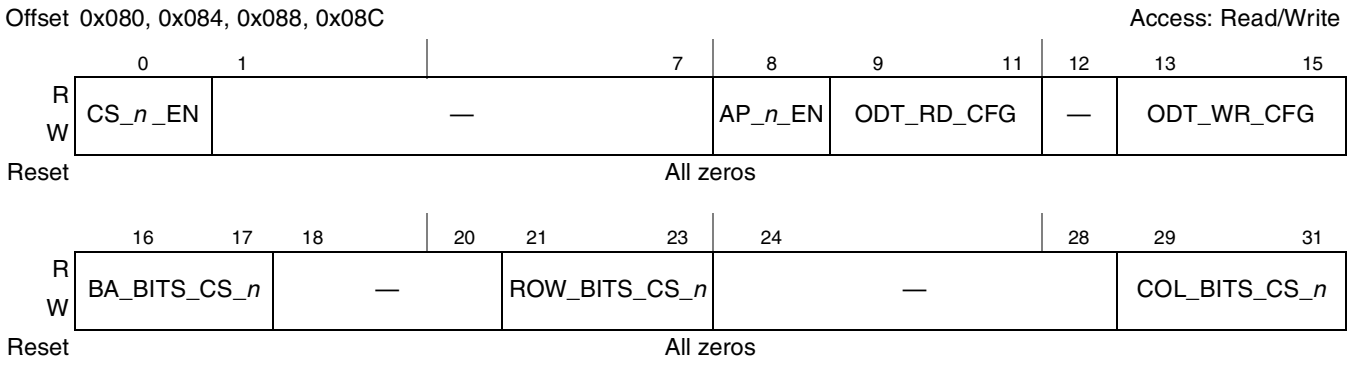


Figure 9-3. Chip Select Configuration Register (CS*n*_CONFIG)

Table 9-7 describes the CS*n*_CONFIG register fields.

Table 9-7. CS*n*_CONFIG Field Descriptions

Bits	Name	Description
0	CS_*n*_EN	Chip select *n* enable 0 Chip select *n* is not active 1 Chip select *n* is active and assumes the state set in CS*n*_BNDS.
1–7	—	Reserved
8	AP_*n*_EN	Chip select *n* auto-precharge enable 0 Chip select *n* will only be auto-precharged if global auto-precharge mode is enabled (DDR_SDRAM_INTERVAL[BSTOPRE] = 0). 1 Chip select *n* will always issue an auto-precharge for read and write transactions.
9–11	ODT_RD_CFG	ODT for reads configuration. Note that CAS latency plus additive latency must be at least 3 cycles for ODT_RD_CFG to be enabled. ODT should only be used with DDR2 memories. 000 Never assert ODT for reads 001 Assert ODT only during reads to CS*n* 010 Assert ODT only during reads to other chip selects 011 Assert ODT only during reads to other DIMM modules. It is assumed that CS0 and CS1 are on the same DIMM module, whereas CS2 and CS3 are on a separate DIMM module. 100 Assert ODT for all reads 101–111 Reserved
12	—	Reserved
13–15	ODT_WR_CFG	ODT for writes configuration. Note that write latency plus additive latency must be at least 3 cycles for ODT _WR_CFG to be enabled. ODT should only be used with DDR2 memories. 000 Never assert ODT for writes 001 Assert ODT only during writes to CS*n* 010 Assert ODT only during writes to other chip selects 011 Assert ODT only during writes to other DIMM modules. It is assumed that CS0 and CS1 are on the same DIMM module, whereas CS2 and CS3 are on a separate DIMM module. 100 Assert ODT for all writes 101–111 Reserved
16–17	BA_BITS_CS_*n*	Number of bank bits for SDRAM on chip select *n*. These bits correspond to the sub-bank bits driven on MBA*n* in Table 9-39 and Table 9-41. 00 2 logical bank bits 01 3 logical bank bits 10–11Reserved
18–20	—	Reserved
21–23	ROW_BITS_CS_*n*	Number of row bits for SDRAM on chip select *n*. See Table 9-39 and Table 9-41 for details. 000 12 row bits 001 13 row bits 010 14 row bits 011 15 row bits 000–111 Reserved

Table 9-7. CS*n*_CONFIG Field Descriptions (continued)

Bits	Name	Description
24–28	—	Reserved
29–31	COL_BITS_CS_*n*	Number of column bits for SDRAM on chip select *n*. For DDR, the decoding is as follows: 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.

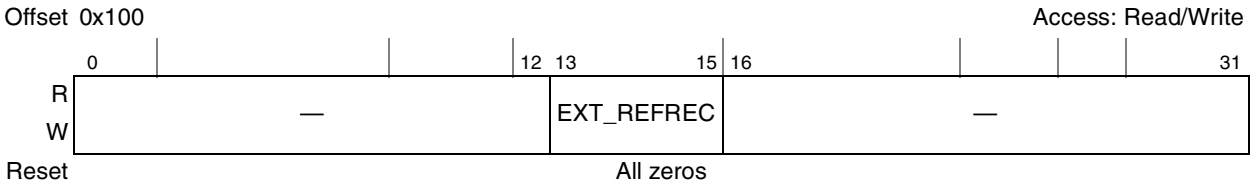


Figure 9-4. DDR SDRAM Timing Configuration 3 (TIMING_CFG_3)

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 || 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.

Offset 0x104 Access: Read/Write

	0 1	2 3	4 5	6 7	8	9 11	12	13 15	16 19	20 23	24 27	28 31
R W	RWT	WRT	RRT	WWT	—	ACT_PD_EXIT	—	PRE_PD_EXIT	—	ODT_PD_EXIT	—	MRS_CYC
Reset	0 0	0 0	0 0	0 0	0	0 0 1	0	0 0 1	0 0 0 0	0 0 0 1	0 0 0 0	0 1 0 1

Figure 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_{RTW}). Specifies how many extra cycles will be added between a read to write turnaround. If 0 clocks is chosen, then the DDR controller will use 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 will determine the read-to-write turnaround as CL – WL + BL/2 + 2. In this equation, CL is the CAS latency rounded up to the next integer, WL is the programmed write latency, and BL is the burst length. 00 0 clocks 10 2 clocks 01 1 clock 11 3 clocks
2–3	WRT	Write-to-read turnaround. Specifies how many extra cycles will be added between a write to read turnaround. If 0 clocks is chosen, then the DDR controller will use 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 will determine the write-to-read turnaround as WL – CL + BL/2 + 1. In this equation, CL is the CAS latency rounded down to the next integer, WL is the programmed write latency, and BL is the burst length. 00 0 clocks 10 2 clocks 01 1 clock 11 3 clocks
4–5	RRT	Read-to-read turnaround. Specifies how many extra cycles will be added between reads to different chip selects. As a default, 3 cycles will be 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 will be the default. Note that DDR2 does not support 8-beat bursts. 00 0 clocks 10 2 clocks 01 1 clock 11 3 clocks
6–7	WWT	Write-to-write turnaround. Specifies how many extra cycles will be added between writes to different chip selects. As a default, 2 cycles will be 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 will be the default. Note that DDR2 does not support 8-beat bursts. 00 0 clocks 10 2 clocks 01 1 clock 11 3 clocks
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.

Offset 0x108 Access: Read/Write

	0	1–3	4–7	8	9–11	12–15	16–19	20	21–23	24	25–27	28	29–31
R W	—	PRETOACT	ACTTOPRE	—	ACTTORW	CASLAT	REFREC	—	WRREC	—	ACTTOACT	—	WRTORD
Reset	All zeros												

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 0001 17 clocks 0010 18 clocks 0011 19 clocks 0100 4 clocks 0101 5 clocks 0110 6 clocks 0111 7 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	$\overline{\text{MCAS}}$ latency from READ command. Number of clock cycles between registration of a READ command by the SDRAM and the availability of the first output data. If a READ command is registered at clock edge n and the latency is m clocks, data is available nominally coincident with clock edge $n + m$. This value must be programmed at initialization as described in Section 9.4.1.8, "DDR SDRAM Control Configuration 2 (DDR_SDRAM_CFG_2).") 0000 Reserved — 1000 4.5 clocks 0001 1 clock — 1001 5 clocks 0010 1.5 clocks — 1010 5.5 clocks 0011 2 clocks — 1011 6 clocks 0100 2.5 clocks — 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	Refresh recovery time (t_{RFC}). Controls the number of clock cycles from a refresh command until an activate command is allowed. This field is concatenated with TIMING_CFG_3[EXTREFREC] to obtain a 7-bit value for the total refresh recovery. Note that hardware adds an additional 8 clock cycles to the final, 7-bit value of the refresh recovery, such that t_{RFC} is calculated as follows: t_{RFC} = {EXT_REFREC || REFREC} + 8. 0000 8 clocks — 0011 11 clocks 0001 9 clocks — ... 0010 10 clocks — 1111 23 clocks
20	—	Reserved, should be cleared.
21–23	WRREC	Last data to precharge minimum interval (t_{WR}). Determines the number of clock cycles from the last data associated with a write command until a precharge command is allowed. 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	Activate-to-activate interval (t_{RRD}). Number of clock cycles from an activate command until another activate command is allowed for a different logical bank in the same physical bank (chip select). 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	Last write data pair to read command issue interval (t_{WTR}). Number of clock cycles between the last write data pair and the subsequent read command to the same physical bank. 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.

Offset 0x10C Access: Read/Write

	0	1–3	4–8	9	10–12	13–15	16–18	19–21	22	23–25	26–31
R	—	ADD_LAT	CPO	—	WR_LAT	—	RD_TO_PRE	WR_DATA_DELAY	—	CKE_PLS	FOUR_ACT
W											
Reset	All zeros										

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]. (DDR2-specific) 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[1]	$\overline{\text{MCAS}}$-to-preamble override. Defines the number of DRAM cycles between when a read is issued and when the corresponding DQS preamble is valid for the memory controller. For these decodings, “READ_LAT” is equal to the CAS latency plus the additive latency. 00000 READ_LAT + 1 00001 Reserved 00010 READ_LAT 00011 READ_LAT + 1/4 00100 READ_LAT + 1/2 00101 READ_LAT + 3/4 00110 READ_LAT + 1 00111 READ_LAT + 5/4 01000 READ_LAT + 3/2 01001 READ_LAT + 7/4 01010 READ_LAT + 2 01011 READ_LAT + 9/4 01100 READ_LAT + 5/2 01101 READ_LAT + 11/4 01110 READ_LAT + 3 01111 READ_LAT + 13/4 10000 READ_LAT + 7/2 10001 READ_LAT + 15/4 10010 READ_LAT + 4 10011 READ_LAT + 17/4 10100 READ_LAT + 9/2 10101 READ_LAT + 19/4 10110–11110 Reserved 11111 Automatic calibration (recommended)
9	—	Reserved

Table 9-11. TIMING_CFG_2 Field Descriptions (continued)

Bits	Name	Description
10–12	WR_LAT	Write latency. Note that the total write latency for DDR2 is equal to WR_LAT + ADD_LAT; the write latency for DDR1 is 1. 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
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. For DDR1 with burst length of 4, must be set to 010; for DDR1 with burst length of 8, must be set to 100. 000 Reserved 001 1 cycle 010 2 cycles 011 3 cycles 100 4 cycles 101–111 Reserved
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 001 1/4 clock delay 010 1/2 clock delay 011 3/4 clock delay 100 1 clock delay 101 5/4 clock delay 110 3/2 clock delay 111 Reserved
22	—	Reserved
23–25	CKE_PLS	Minimum CKE pulse width (t_{CKE}). Can be set to 001 for DDR1. 000 Reserved 001 1 cycle 010 2 cycles 011 3 cycles 100 4 cycles 101–111 Reserved
26–31	FOUR_ACT	Window for four activates (t_{FAW}). This is applied to DDR2 with eight logical banks only. Must be set to 000001 for DDR1. This field is concatenated with TIMING_CFG_3[EXT_FOUR_ACT] to obtain a 5-bit value for the total activate to precharge time. Note that the decode of 000000–000011 is equal to 16–19 clocks when TIMING_CFG_3[EXT_ACTTOPRE] = 0, but it is equal to 0–3 clocks when TIMING_CFG_3[EXT_ACTTOPRE] = 1. 000000 Reserved 000001 1 cycle 000010 2 cycles 000011 3 cycles 000100 4 cycles ... 010011 19 cycles 010100 20 cycles 010101–111111 Reserved

[1] For CPO decodings other than 00000 and 11111, ‘READ_LAT’ is rounded up to the next integer value.

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.

Offset 0x110 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MEM_EN	SREN	ECC_EN	RD_EN	—	SDRAM_TYPE			—		DYN_PWR	—	32_BE	8_BE	NCAP	—
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	2T_EN	BA_INTLV_CTL							—		x32_EN	PCHB8	HSE	—	MEM_HALT	BI
Reset	All zeros															

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 DIMM enable. Specifies the type of DIMM used in the system. 0 Indicates unbuffered DIMMs. 1 Indicates registered DIMMs.
4	—	Reserved
5–7	SDRAM_TYPE	Type of SDRAM device to be used. This field will be used when issuing the automatic hardware initialization sequence to DRAM via Mode Register Set and Extended Mode Register Set commands. Default value is 010 designating DDR1 SDRAM. 000–001Reserved 010 DDR1 SDRAM 011 DDR2 SDRAM 100 Reserved 101 Reserved 110 Reserved 111 Reserved
8–9	—	Reserved

Table 9-12. DDR_SDRAM_CFG Field Descriptions (continued)

Bits	Name	Description
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	—	Reserved
12	32_BE	32-bit bus enable. 0 64-bit bus is used. 1 32-bit bus is used.
13	8_BE	8-beat burst enable. 8-beat bursts may only be used with DDR1 SDRAM type; must be cleared for DDR2. 0 4-beat bursts are used on the DRAM interface. 1 8-beat bursts are used on the DRAM interface (only use with 32_BE = 1).
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 will need 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.
17–23	BA_INTLV_CTL	Bank (chip select) interleaving control. Set this field only if you wish to use bank interleaving. ('*x*' denotes a don't care bit value. All unlisted field values are reserved.) 0000000 No external memory banks are interleaved 1000000 External memory banks 0 and 1 are interleaved 0100000 External memory banks 2 and 3 are interleaved 1100000 External memory banks 0 and 1 are interleaved together **and** banks 2 and 3 are interleaved together *xx*00100 External memory banks 0 through 3 are all interleaved together
24–25	—	Reserved
26	x32_EN	x32 enable. 0 Either x8 or x16 discrete DRAM chips are used. In this mode, each data byte has a dedicated corresponding data strobe. 1 x32 discrete DRAM chips are used. In this mode, DQS0 will be used to capture DQ[0:31], DQS4 will be used to capture DQ[32:63] and DQS8 will be used to capture ECC[0:7].
27	PCHB8	Precharge bit 8 enable. 0 MA[10] will be used to indicate the auto-precharge and precharge all commands. 1 MA[8] will be used to indicate the auto-precharge and precharge all commands. If x32_EN is cleared, then PCHB8 should be cleared as well.
28	HSE	Global half-strength override Sets I/O driver impedance to half strength. This impedance will be used by the MDIC, address/command, data, and clock impedance values, but only if automatic hardware calibration is disabled and the corresponding group's software override is disabled in the DDR control driver register(s) described in Section 5.3.2.8, "DDR Control Driver Register (DDRCDR)." 0 I/O driver impedance will be configured to full strength. 1 I/O driver impedance will be configured to half strength.

Table 9-12. DDR_SDRAM_CFG Field Descriptions (continued)

Bits	Name	Description
29	—	Reserved
30	MEM_HALT	DDR memory controller halt. When this bit is set, the memory controller will 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 will accept new transactions. 1 DDR controller will finish any remaining transactions, and then it will remain halted until this bit is cleared by software.
31	BI	Bypass initialization 0 DDR controller will cycle through initialization routine based on SDRAM_TYPE 1 Initialization routine will be 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. See Section 9.4.1.15, “DDR Initialization Address (DDR_INIT_ADDR),” 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.

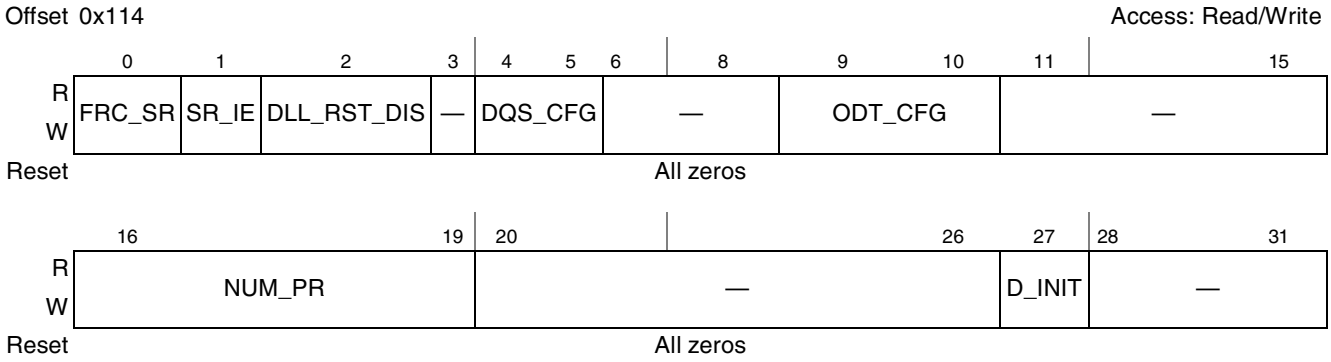


Figure 9-9. DDR SDRAM Control Configuration Register 2 (DDR_SDRAM_CFG_2)

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 will operate in normal mode. 1 DDR controller will enter self-refresh mode.
1	SR_IE	Self-refresh interrupt enable. The DDR controller can be placed into self refresh mode if DDR_SR_REQ (IRQ1) is asserted. This is considered a ‘panic interrupt’ by the DDR controller, and it will enter self refresh as soon as possible. DDR_SDRAM_CFG[SREN] must also be set if the panic interrupt will be used. 0 DDR controller will not enter self-refresh mode if panic interrupt is asserted. 1 DDR controller will enter self-refresh mode if panic interrupt is asserted.

Table 9-13. DDR_SDRAM_CFG_2 Field Descriptions (continued)

Bits	Name	Description
2	DLL_RST_DIS	DLL reset disable. The DDR controller will typically issue 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 will issue a DLL reset to the DRAMs when exiting self refresh. 1 DDR controller will 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 Differential DQS signals are used for DDR2 support. 10 Reserved 11 Reserved
6–8	—	Reserved
9–10	ODT_CFG	ODT configuration. This field defines how ODT will be driven to the on-chip IOs. See Section 5.3.2.8, "DDR Control Driver Register (DDRCDR)," which defines the termination value that will be used. (DDR2-specific, must be cleared for DDR1) 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
16–19	NUM_PR	Number of posted refreshes. This will determine 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. For example, some DDR1 SDRAMs will not be able to use more than 3 posted refreshes because the required refresh interval could then exceed the maximum constraint for t_{ras}. 0000 Reserved 0001 1 refresh will be issued at a time 0010 2 refreshes will be issued at a time 0011 3 refreshes will be issued at a time ... 1000 8 refreshes will be issued at a time 1001–1111Reserved
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 will automatically initialize DRAM after it is enabled. This bit will be automatically cleared by hardware once the initialization is completed. 0 There is not data initialization in progress, and no data initialization is scheduled 1 The memory controller will initialize memory once it is enabled. This bit will remain asserted until the initialization is complete. The value in DDR_DATA_INIT register will be 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.

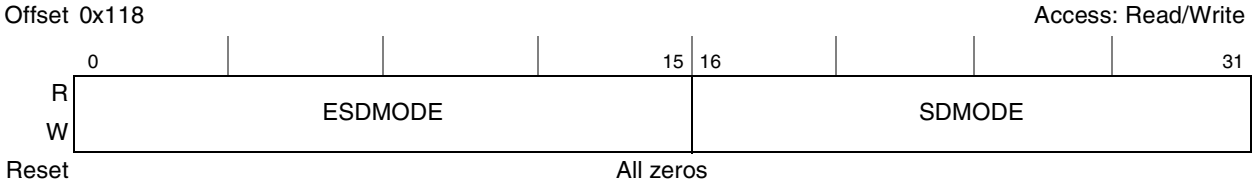


Figure 9-10. DDR SDRAM Mode Configuration Register (DDR_SDRAM_MODE)

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 upon 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).

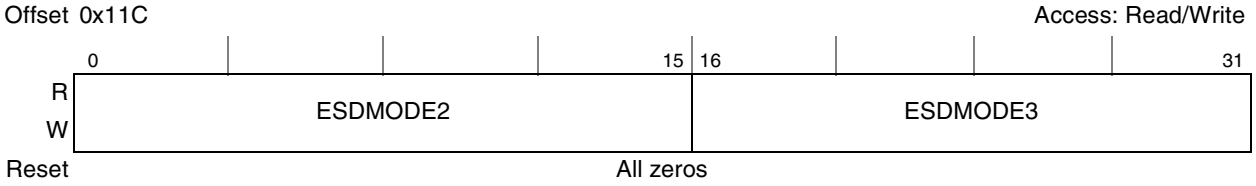


Figure 9-11. DDR SDRAM Mode 2 Configuration Register (DDR_SDRAM_MODE_2)

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.

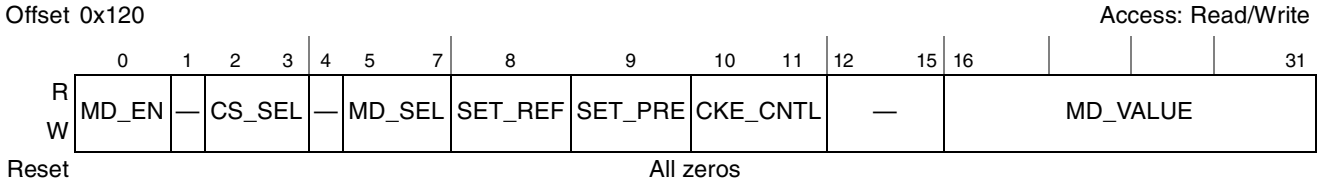


Figure 9-12. DDR SDRAM Mode Control Register (DDR_SDRAM_MD_CNTL)

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: • MODE REGISTER SET • EXTENDED MODE REGISTER SET • EXTENDED MODE REGISTER SET 2 • EXTENDED MODE REGISTER SET 3 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 will be 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 Chip select 2 is active 11 Chip select 3 is active
4	—	Reserved
5–7	MD_SEL	Mode register select. MD_SEL specifies one of the following: • During a mode select command, selects the SDRAM mode register to be changed • During a precharge command, selects the SDRAM logical bank to be precharged. A precharge all command ignores this field. • During a refresh command, this field is ignored. Note that MD_SEL contains the value that will be presented onto the memory bank address pins (MBA*n*) of the DDR controller. 000 MR 001 EMR 010 EMR2 011 EMR3
8	SET_REF	Set refresh. Forces an immediate refresh to be issued to the chip select specified by DDR_SDRAM_MD_CNTL[CS_SEL]. This bit will be set by software and cleared by hardware once the command has been issued. 0 Indicates that no refresh command needs to be issued. 1 Indicates that a refresh command is ready to be issued.
9	SET_PRE	Set precharge. Forces a precharge or precharge all to be issued to the chip select specified by DDR_SDRAM_MD_CNTL[CS_SEL]. This bit will be set by software and cleared by hardware once the command has been issued. 0 Indicates that no precharge all command needs to be issued. 1 Indicates that a precharge all command is ready to be issued.
10–11	CKE_CNTL	Clock enable control. Allows software to globally clear or set all CKE signals issued to DRAM. Once software has forced the value driven on CKE, that value will continue to be forced until software clears the CKE_CNTL bits. At that time, the DDR controller will continue to drive the CKE signals to the same value forced by software until another event causes the CKE signals to change (i.e., self refresh entry/exit, power down entry/exit). 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

Table 9-16. DDR_SDRAM_MD_CNTL Field Descriptions (continued)

Bits	Name	Description
12–15	—	Reserved
16–31	MD_VALUE	Mode register value. This field, which specifies the value that will be 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. 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: 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

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	—
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.

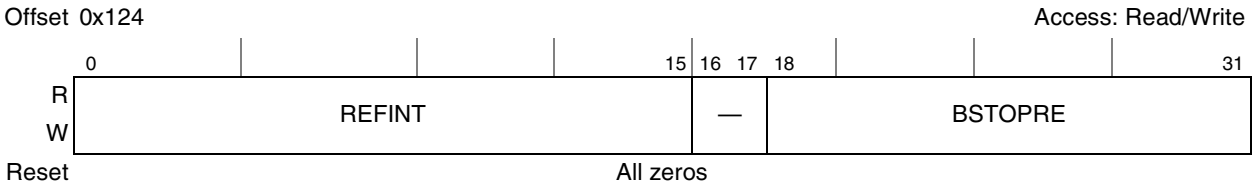


Figure 9-13. DDR SDRAM Interval Configuration Register (DDR_SDRAM_INTERVAL)

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 will not be 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 will be used to initialize memory if DDR_SDRAM_CFG2[D_INIT] is set.

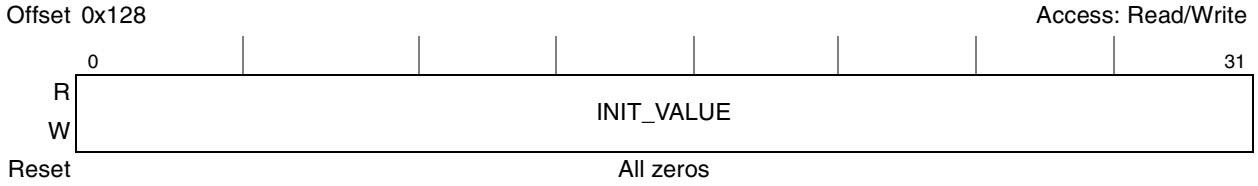


Figure 9-14. DDR SDRAM Data Initialization Configuration Register (DDR_DATA_INIT)

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 will be 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 source-synchronous option, along with a 1/8 cycle clock adjustment.

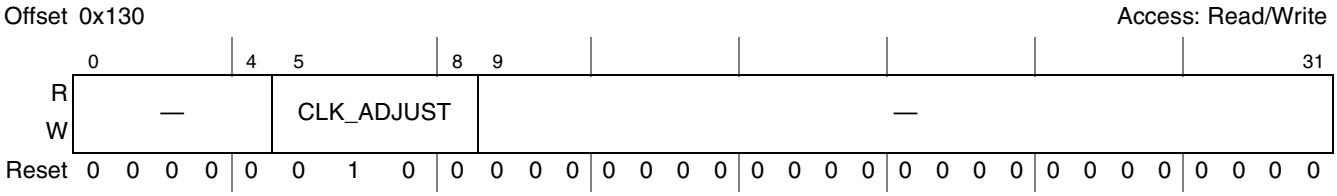


Figure 9-15. DDR SDRAM Clock Control Configuration Register (DDR_SDRAM_CLK_CNTL)

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
5–8	CLK_ADJUST	Clock adjust 0000 Clock will be launched aligned with address/command 0001 Clock will be launched 1/8 applied cycle after address/command 0010 Clock will be launched 1/4 applied cycle after address/command 0011 Clock will be launched 3/8 applied cycle after address/command 0100 Clock will be launched 1/2 applied cycle after address/command 0101 Clock will be launched 5/8 applied cycle after address/command 0110 Clock will be launched 3/4 applied cycle after address/command 0111 Clock will be launched 7/8 applied cycle after address/command 1000 Clock will be launched 1 applied cycle after address/command 1001–1111Reserved
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 will be used for the automatic CAS to preamble calibration after POR.

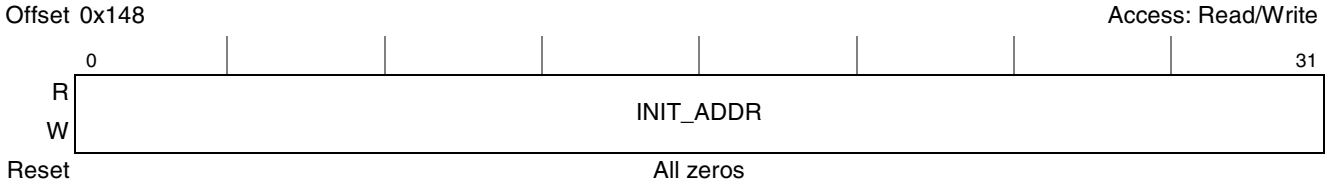


Figure 9-16. DDR Initialization Address Configuration Register (DDR_INIT_ADDR)

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 will be 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.

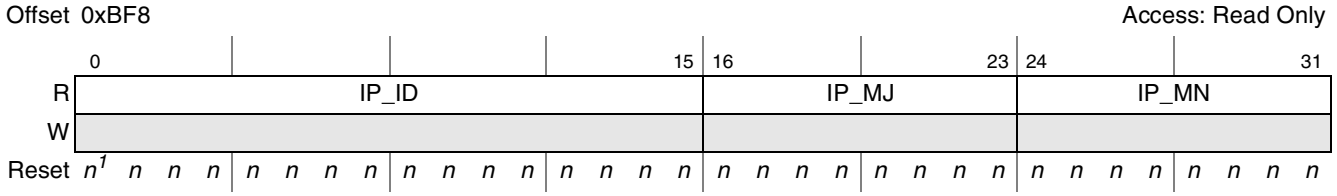


Figure 9-17. DDR IP Block Revision 1 (DDR_IP_REV1)

1 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 0x00.

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.

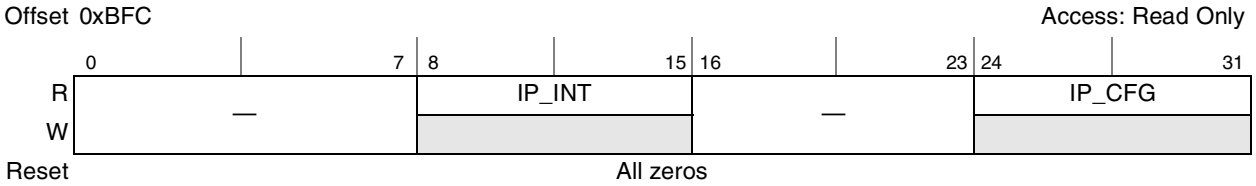


Figure 9-18. DDR IP Block Revision 2 (DDR_IP_REV2)

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
16–23	—	Reserved
24–31	IP_CFG	IP block configuration options

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.

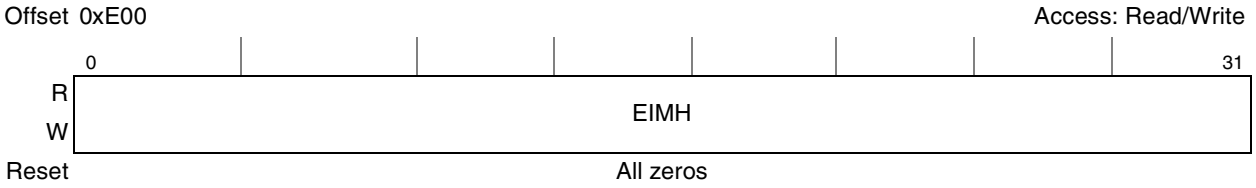


Figure 9-19. Memory Data Path Error Injection Mask High Register (DATA_ERR_INJECT_HI)

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.

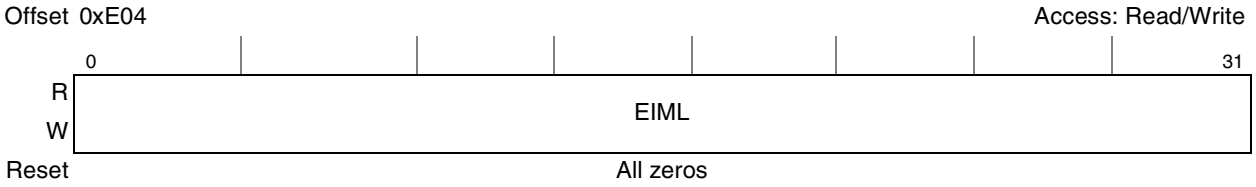


Figure 9-20. Memory Data Path Error Injection Mask Low Register (DATA_ERR_INJECT_LO)

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.

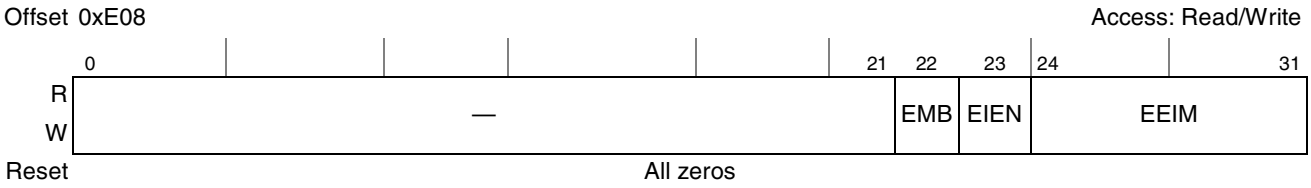


Figure 9-21. Memory Data Path Error Injection Mask ECC Register (ERR_INJECT)

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.
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.

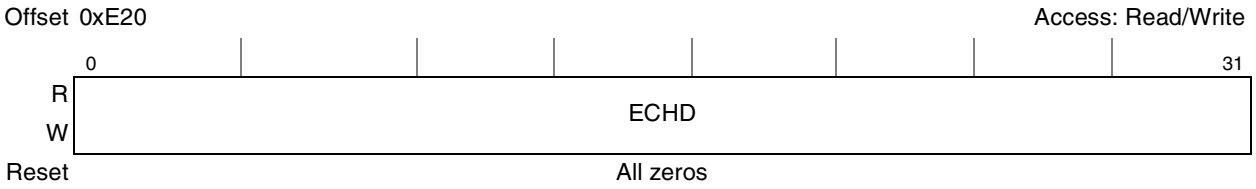


Figure 9-22. Memory Data Path Read Capture High Register (CAPTURE_DATA_HI)

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.

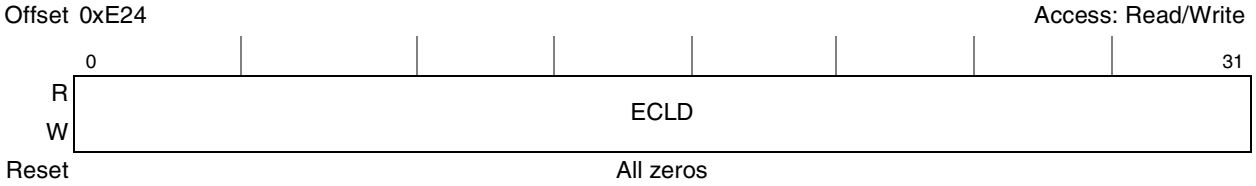


Figure 9-23. Memory Data Path Read Capture Low Register (CAPTURE_DATA_LO)

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.

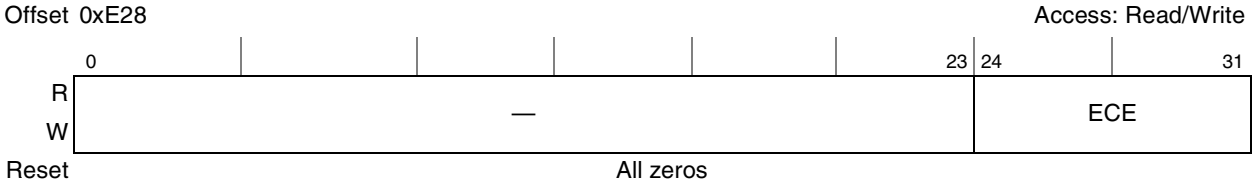


Figure 9-24. Memory Data Path Read Capture ECC Register (CAPTURE_ECC)

Table 9-29 describes the CAPTURE_ECC fields.

Table 9-29. CAPTURE_ECC Field Descriptions

Bits	Name	Description
0–23	—	Reserved
24–31	ECE	Error capture ECC. Captures the ECC bits on the data path whenever errors are detected.

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.

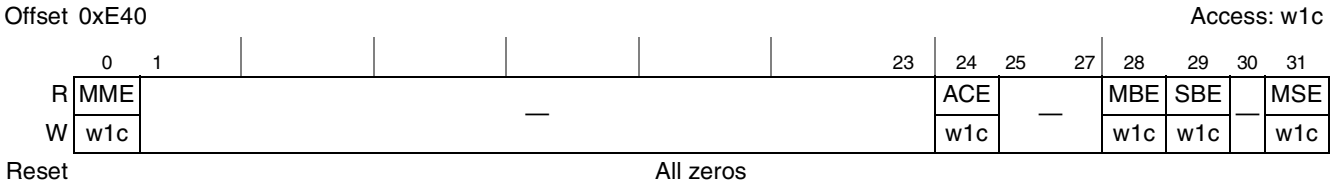


Figure 9-25. Memory Error Detect Register (ERR_DETECT)

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
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.

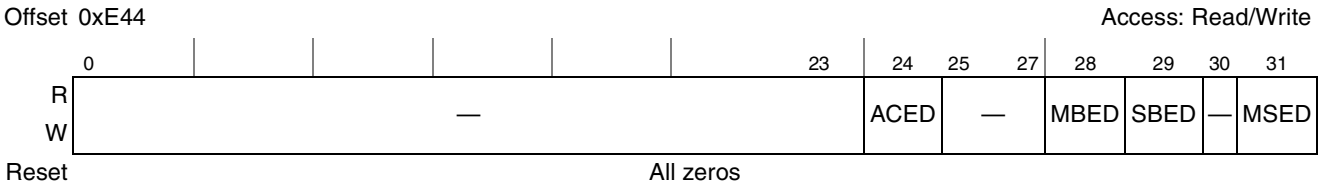


Figure 9-26. Memory Error Disable Register (ERR_DISABLE)

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. See Section 9.5.12, “Error Management,” for more information. 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.
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).

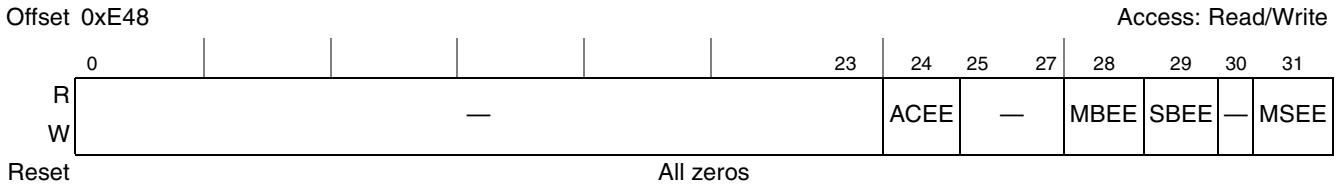


Figure 9-27. Memory Error Interrupt Enable Register (ERR_INT_EN)

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. See Section 9.5.12, “Error Management,” for more information. 0 Multiple-bit ECC errors cannot generate interrupts. 1 Multiple-bit ECC errors generate interrupts.

Table 9-32. ERR_INT_EN Field Descriptions (continued)

Bits	Name	Description
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.

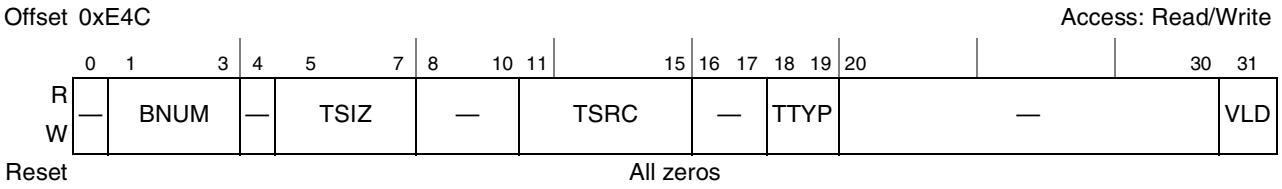


Figure 9-28. Memory Error Attributes Capture Register (CAPTURE_ATTRIBUTES)

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.
8–10	—	Reserved
11–15	TSRC	Transaction source for the error 00000 e300 core data transaction 00001 Reserved 00010 e300 core instruction fetch 00011 Reserved 00100 TSEC 1 00101 TSEC 2 00110 USB MPH 00111 USB DR 01000 Encryption core 01001 I^2C (boot sequencer) 01010 JTAG 01011 Reserved 01100 Reserved 01101 PCI 1 01110 PCI 2 01111 DMA 10000–11111 Reserved
16–17	—	Reserved

Table 9-33. CAPTURE_ATTRIBUTES Field Descriptions (continued)

Bits	Name	Description
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.

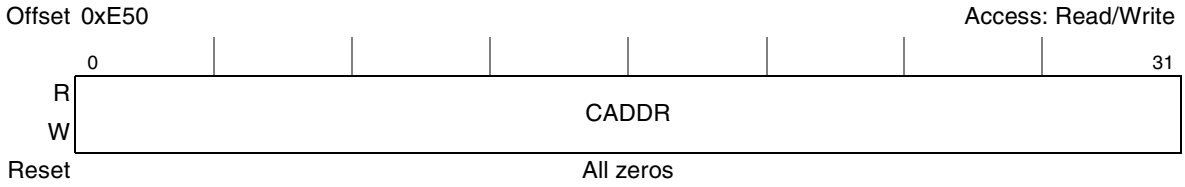


Figure 9-29. Memory Error Address Capture Register (CAPTURE_ADDRESS)

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.

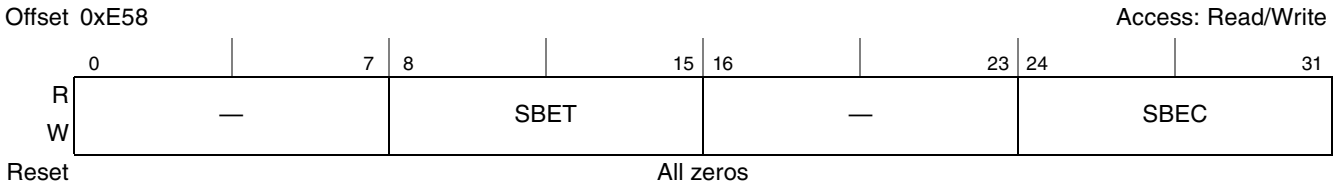


Figure 9-30. Single-Bit ECC Memory Error Management Register (ERR_SBE)

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.
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 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 and DDR SDRAMs. The memory system allows a wide range of memory devices to be mapped to any arbitrary chip select, and support is provided for registered DIMMs and unbuffered DIMMs. However, registered DIMMs cannot be mixed with unbuffered DIMMs.

Figure 9-31 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.

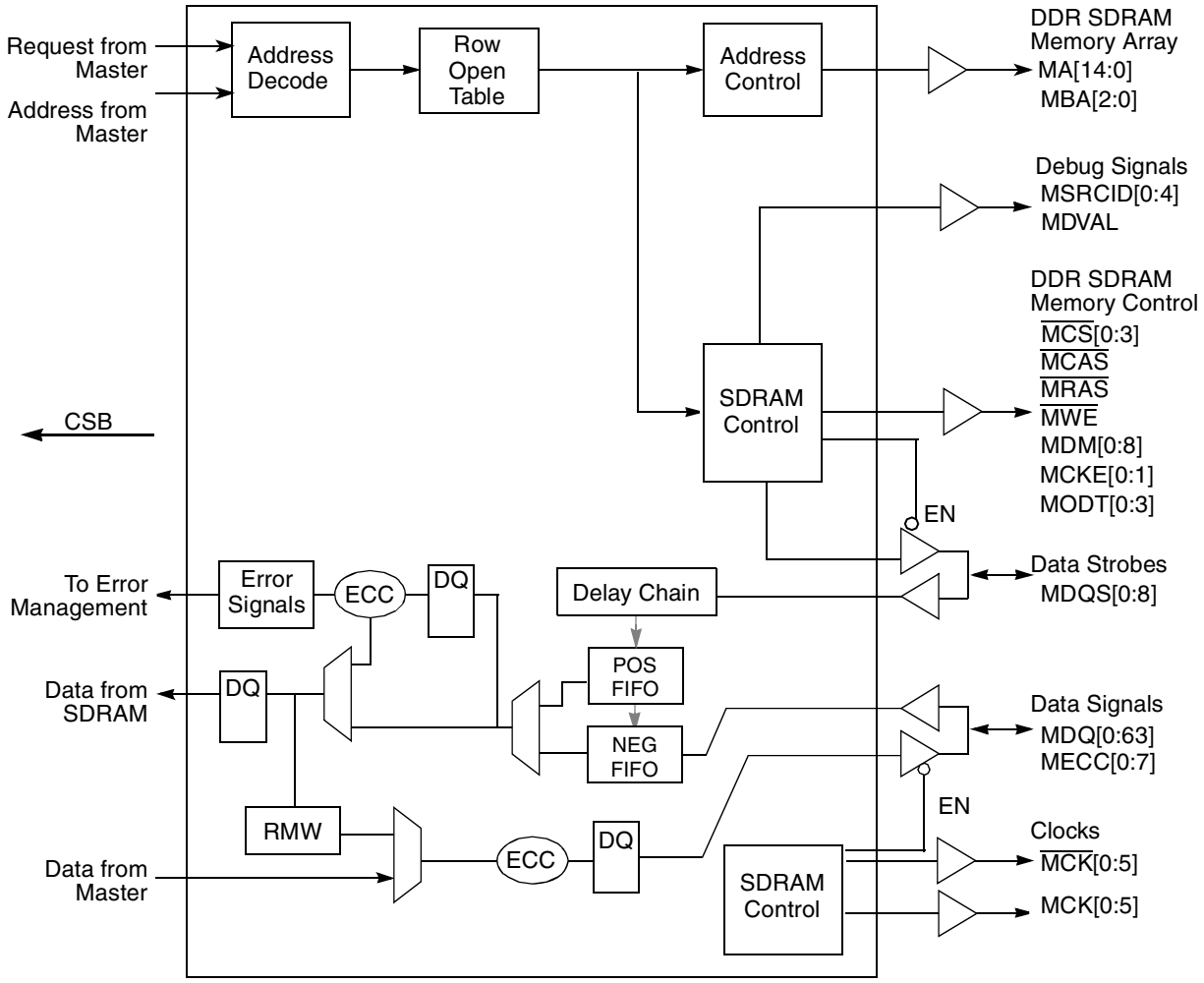


Figure 9-31. DDR Memory Controller Block Diagram

The memory interface supports as many as four physical banks of 64/72-bit-wide or 32/40-bit wide memory. Bank sizes up to 2 Gbytes (maximum total physical memory size of 4 Gbytes) are supported, providing up to a maximum of 4 Gbytes of DDR main memory.

Programmable parameters allow for a variety of memory organizations and timings. Optional error checking and correcting (ECC) protection is provided for the DDR SDRAM data bus. Using ECC, the DDR memory controller detects and corrects all single-bit errors within the 64-bit or 32-bit data bus, detects all double-bit errors within the 64-bit or 32-bit data bus, and detects all errors within a nibble. The controller allows as many as 32 pages to be open simultaneously. The amount of time (in clock cycles) the pages remain open is programmable with DDR_SDRAM_INTERVAL[BSTOPRE].

Read and write accesses to memory are burst oriented; accesses start at a selected location and continue for a programmed number of higher locations (4 or 8) in a programmed sequence. Accesses to closed pages start with the registration of an ACTIVE command followed by a READ or WRITE. (Accessing open pages does not require an ACTIVE command.) The address bits registered coincident with the activate command specifies the logical bank and row to be accessed. The address coincident with the READ or WRITE command specify the logical bank and starting column for the burst access.

The data interface is source synchronous, meaning whatever sources the data also provides a clocking signal to synchronize data reception. These bidirectional data strobes (MDQS[0:8]) are inputs to the controller during reads and outputs during writes. The DDR SDRAM specification requires the data strobe signals to be centered within the data tenure during writes and to be offset by the controller to the center of the data tenure during reads. This delay is implemented in the controller for both reads and writes.

When ECC is enabled, 1 clock cycle is added to the read path to check ECC and correct single-bit errors. ECC generation does not add a cycle to the write path.

The address and command interface is also source synchronous, although 1/8 cycle adjustments are provided for adjusting the clock alignment.

Figure 9-32 shows an example DDR SDRAM configuration with four logical banks.

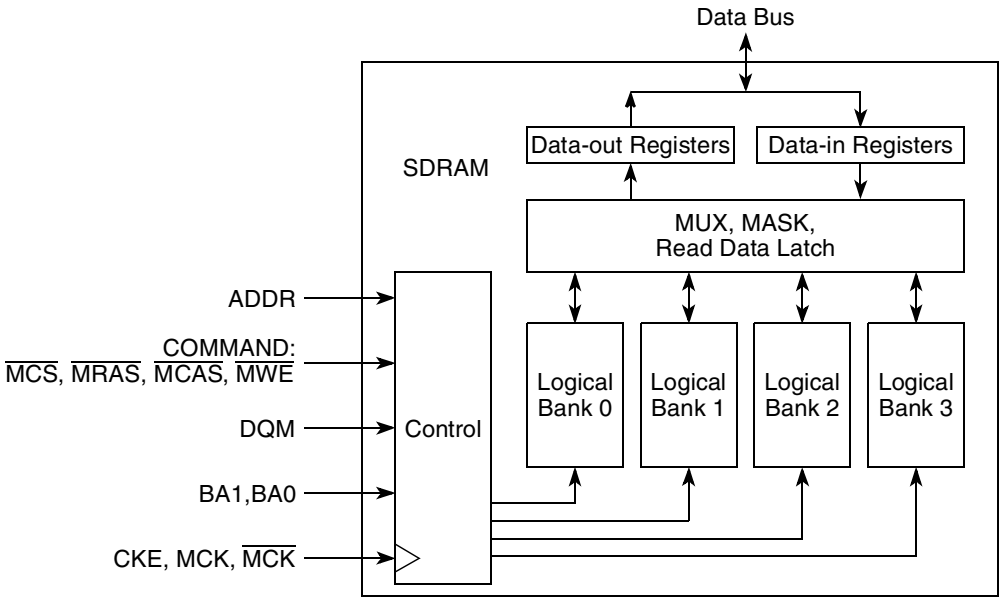


Figure 9-32. Typical Dual Data Rate SDRAM Internal Organization

Figure 9-33 shows some typical signal connections.

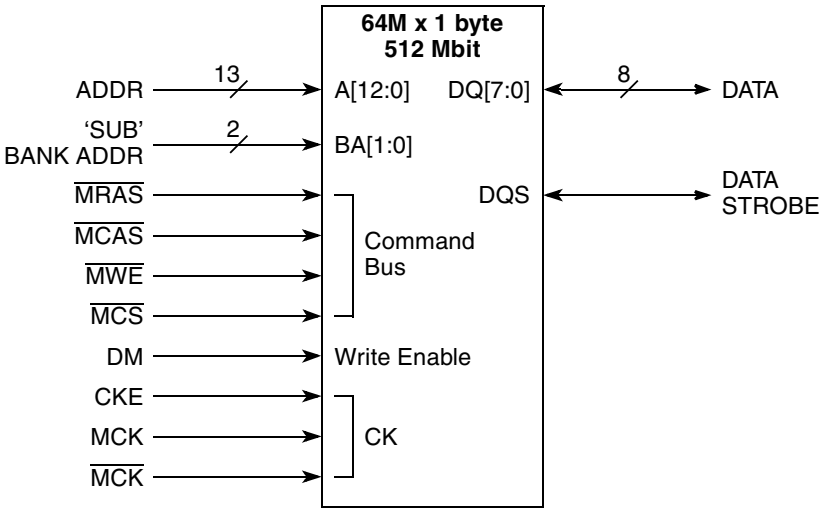
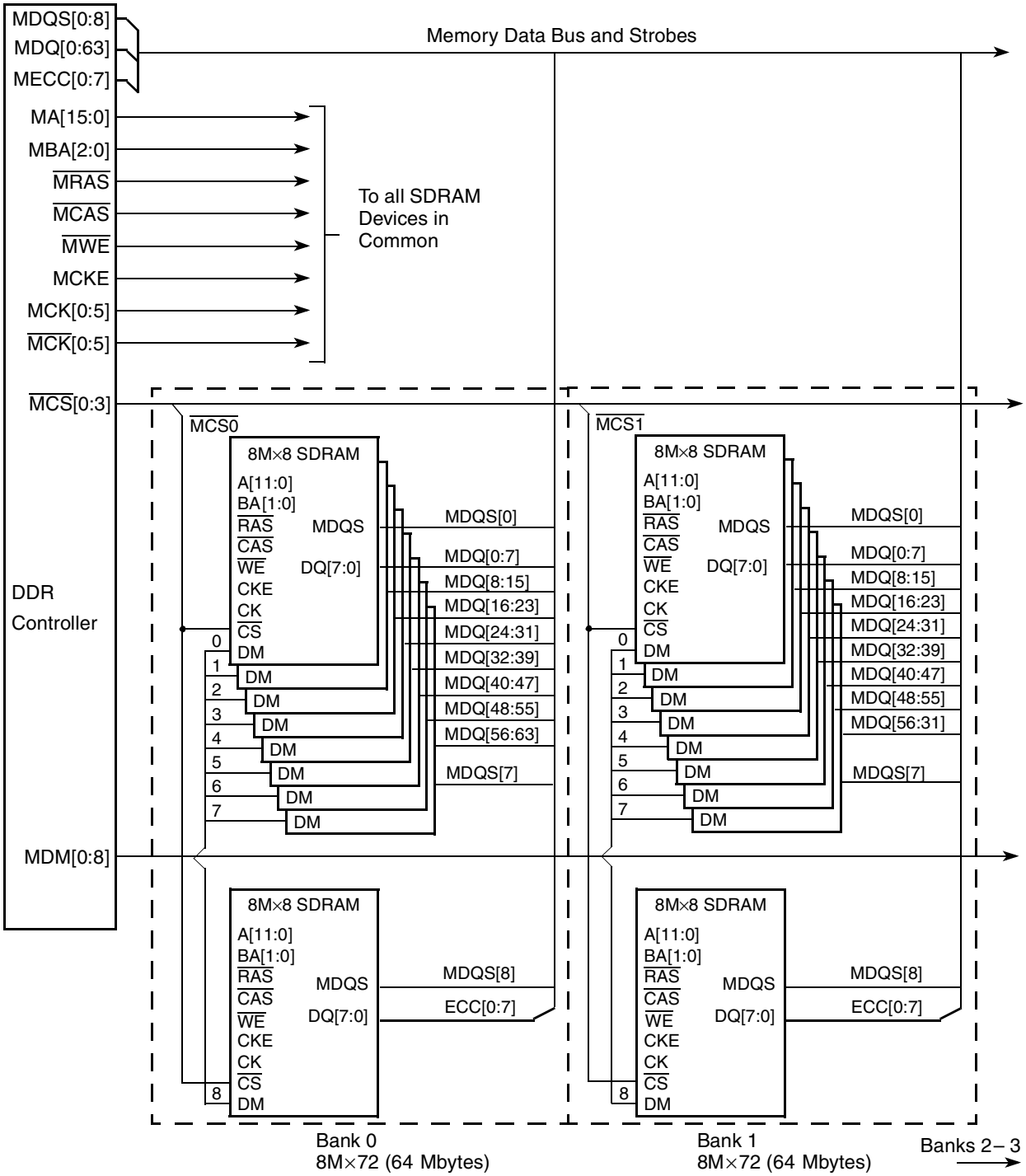


Figure 9-33. Typical DDR SDRAM Interface Signals

Figure 9-34 shows an example DDR SDRAM configuration with four physical banks each comprised of nine 8M × 8 DDR modules for a total of 256 Mbytes of system memory. One of the nine modules is used for the memory's ECC checking function. Certain address and control lines may require buffering. Analysis of the device's AC timing specifications, desired memory operating frequency, capacitive loads, and board routing loads can assist the system designer in deciding signal buffering requirements. The DDR memory controller drives 15 address pins, but in this example the DDR SDRAM devices use only 12 bits.



1. All signals are connected in common (in parallel) except for MCS[0:3], MCK[0:5], MDM[0:8], and the data bus signals.
2. Each of the MCS[0:3] signals correspond with a separate physical bank of memory.
3. Buffering may be needed if large memory arrays are used.
4. MCK[0:5] may be apportioned among all memory devices. Complementary bus is not shown.

Figure 9-34. Example 256-Mbyte DDR SDRAM Configuration With ECC

Section 9.5.12, "Error Management," explains how the DDR memory controller handles errors.

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. Fifteen multiplexed address signals and three logical bank select signals support device densities from 64 Mbit to 4 Gbits. Four chip select ($\overline{CS}$) signals support up to two DIMMs of memory. The DDR SDRAM physical banks can be built from standard memory modules or directly-attached memory devices. The data path to individual physical banks is 64 or 32 bits wide, 72 or 40 bits with ECC. The DDR memory controller supports physical bank sizes from 16 Mbytes to 4 Gbytes. The physical banks can be constructed using x8, x16, or x32 memory devices. The memory technologies supported are 64 Mbit, 128 Mbit, 256 Mbit, 512 Mbit, 1 Gbit, 2 Gbit and 4 Gbit. 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 lane 0–7, MDM[0:7], MDQS[0:7], and MDQ[0:63] when DDR SDRAM memories are used with x8 or x16 devices.

Table 9-36. Byte Lane to Data Relationship

Data Byte Lane	Data Bus Mask	Data Bus Strobe	Data Bus 64-Bit Mode
0 (MSB)	MDM[0]	MDQS[0]	MDQ[0:7]
1	MDM[1]	MDQS[1]	MDQ[8:15]
2	MDM[2]	MDQS[2]	MDQ[16:23]
3	MDM[3]	MDQS[3]	MDQ[24:31]
4	MDM[4]	MDQS[4]	MDQ[32:39]
5	MDM[5]	MDQS[5]	MDQ[40:47]
6	MDM[6]	MDQS[6]	MDQ[48:55]
7 (LSB)	MDM[7]	MDQS[7]	MDQ[56:63]

9.5.1.1 Supported DDR SDRAM Organizations

Although the DDR memory controller multiplexes row and column address bits onto 15 memory address signals and 3 logical bank select signals, a physical bank may be implemented with memory devices requiring fewer than 31 address bits. The physical bank may be configured to provide from 12 to 15 row address bits, plus 2 or 3 logical bank-select bits and from 8–11 column address bits.

Table 9-37 describes DDR SDRAM device configurations supported by the DDR memory controller.

NOTE

DDR SDRAM is limited to 30 total address bits.

Table 9-37. Supported DDR1 SDRAM Device Configurations

SDRAM Device	Device Configuration	Row x Column x Sub-bank Bits	64-Bit Bank Size	Four Banks of Memory
64 Mbits	8 Mbits x 8	12 x 9 x 2	64 Mbytes	256 Mbytes
64 Mbits	4 Mbits x 16	12 x 8 x 2	32 Mbytes	128 Mbytes
128 Mbits	16 Mbits x 8	12 x 10 x 2	128 Mbytes	512 Mbytes
128 Mbits	8 Mbits x 16	12 x 9 x 2	64 Mbytes	256 Mbytes
256 Mbits	32 Mbits x 8	13 x 10 x 2	256 Mbytes	1 Gbytes
256 Mbits	16 Mbits x 16	13 x 9 x 2	128 Mbytes	512 Mbytes
512 Mbits	64 Mbits x 8	13 x 11 x 2	512 Mbytes	2 Gbytes
512 Mbits	32 Mbits x 16	13 x 10 x 2	256 Mbytes	1 Gbytes
1 Gbits	128 Mbits x 8	14 x 11 x 2	1 Gbyte	4 Gbytes
1 Gbits	64 Mbits x 16	14 x 10 x 2	512 Mbytes	2 Gbytes
2 Gbits	256 Mbits x 8	15 x 11 x 2	2 Gbytes	4 Gbytes (two banks)
2 Gbits	128 Mbits x 16	15 x 10 x 2	1 Gbytes	4 Gbytes

Table 9-38. Supported DDR2 SDRAM Device Configurations

SDRAM Device	Device Configuration	Row x Column x Sub-bank Bits	64-Bit Bank Size	Four Banks of Memory
256 Mbits	32 Mbits x 8	13 x 10 x 2	256 Mbytes	1 Gbytes
256 Mbits	16 Mbits x 16	13 x 9 x 2	128 Mbytes	512 Mbytes
512 Mbits	64 Mbits x 8	14 x 10 x 2	512 Mbytes	2 Gbytes
512 Mbits	32 Mbits x 16	13 x 10 x 2	256 Mbytes	1 Gbytes
1 Gbits	128 Mbits x 8	14 x 10 x 3	1 Gbyte	4 Gbytes
1 Gbits	64 Mbits x 16	13 x 10 x 3	512 Mbytes	2 Gbytes
2 Gbits	256 Mbits x 8	15 x 10 x 3	2 Gbytes	4 Gbytes (two banks)
2 Gbits	128Mbits x 16	14 x 10 x 3	1 Gbyte	4 Gbytes
4 Gbits	256 Mbits x 16	15 x 10 x 3	2 Gbytes	4 Gbytes (two banks)

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.”

By using a memory-polling algorithm at power-on reset or by querying the JEDEC serial presence detect capability of memory modules, system firmware uses the memory-boundary registers to configure the DDR memory controller to map the size of each bank in memory. The memory controller uses its bank map to assert the appropriate $\overline{\text{MCS}}n$ signal for memory accesses according to the provided bank starting and ending addresses. The memory banks are not required to be mapped to a contiguous address space.

9.5.2 DDR SDRAM Address Multiplexing

Table 9-39, Table 9-40, Table 9-41, and Table 9-42 show the address bit encodings for each DDR SDRAM configuration. The address presented at the memory controller signals MA[14:0] use MA[14] as the msb and MA[0] as the lsb. Also, MA[10] is used as the auto-precharge bit in DDR1/DDR2 modes for reads and writes, so the column address can never use MA[10].

Table 9-39. DDR1 Address Multiplexing for 64-bit Data Bus with Interleaving Disabled

Row x 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	29–31
15 x 11 x 2	$\overline{\text{MRAS}}$		14	13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																	1	0												
	$\overline{\text{MCAS}}$																			11	9	8	7	6	5	4	3	2	1	0	
15 x 10 x 2	$\overline{\text{MRAS}}$			14	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	
14 x 11 x 2	$\overline{\text{MRAS}}$			13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																	1	0												
	$\overline{\text{MCAS}}$																			11	9	8	7	6	5	4	3	2	1	0	
14 x 10 x 2	$\overline{\text{MRAS}}$				13	12	11	10	9	8	7	6	5	4	3	2	1	0													
	MBA																		1	0											
	$\overline{\text{MCAS}}$																				9	8	7	6	5	4	3	2	1	0	
13 x 11 x 2	$\overline{\text{MRAS}}$				12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																	1	0												
	$\overline{\text{MCAS}}$																			11	9	8	7	6	5	4	3	2	1	0	
13 x 10 x 2	$\overline{\text{MRAS}}$					12	11	10	9	8	7	6	5	4	3	2	1	0													
	MBA																		1	0											
	$\overline{\text{MCAS}}$																				9	8	7	6	5	4	3	2	1	0	
13 x 9 x 2	$\overline{\text{MRAS}}$						12	11	10	9	8	7	6	5	4	3	2	1	0												
	MBA																			1	0										
	$\overline{\text{MCAS}}$																					8	7	6	5	4	3	2	1	0	
12 x 10 x 2	$\overline{\text{MRAS}}$						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	
12 x 9 x 2	$\overline{\text{MRAS}}$							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	
12 x 8 x 2	$\overline{\text{MRAS}}$								11	10	9	8	7	6	5	4	3	2	1	0											
	MBA																				1	0									
	$\overline{\text{MCAS}}$																						7	6	5	4	3	2	1	0	

Table 9-40. DDR1 Address Multiplexing for 32-bit Data Bus with Interleaving Disabled

Row x 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
15 x 11 x 2	$\overline{\text{MRAS}}$			14	13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																		1	0											
	$\overline{\text{MCAS}}$																				11	9	8	7	6	5	4	3	2	1	0
15 x 10 x 2	$\overline{\text{MRAS}}$				14	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
14 x 11 x 2	$\overline{\text{MRAS}}$				13	12	11	10	9	8	7	6	5	4	3	2	1	0													
	MBA																		1	0											
	$\overline{\text{MCAS}}$																				11	9	8	7	6	5	4	3	2	1	0
14 x 10 x 2	$\overline{\text{MRAS}}$					13	12	11	10	9	8	7	6	5	4	3	2	1	0												
	MBA																			1	0										
	$\overline{\text{MCAS}}$																					9	8	7	6	5	4	3	2	1	0
13 x 11 x 2	$\overline{\text{MRAS}}$					12	11	10	9	8	7	6	5	4	3	2	1	0													
	MBA																		1	0											
	$\overline{\text{MCAS}}$																				11	9	8	7	6	5	4	3	2	1	0
13 x 10 x 2	$\overline{\text{MRAS}}$						12	11	10	9	8	7	6	5	4	3	2	1	0												
	MBA																			1	0										
	$\overline{\text{MCAS}}$																					9	8	7	6	5	4	3	2	1	0
13 x 9 x 2	$\overline{\text{MRAS}}$							12	11	10	9	8	7	6	5	4	3	2	1	0											
	MBA																				1	0									
	$\overline{\text{MCAS}}$																						8	7	6	5	4	3	2	1	0
12 x 10 x 2	$\overline{\text{MRAS}}$							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
12 x 9 x 2	$\overline{\text{MRAS}}$								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
12 x 8 x 2	$\overline{\text{MRAS}}$									11	10	9	8	7	6	5	4	3	2	1	0										
	MBA																					1	0								
	$\overline{\text{MCAS}}$																							7	6	5	4	3	2	1	0

Table 9-41. DDR2 Address Multiplexing for 64-bit Data Bus with Interleaving Disabled

Row x 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	29–31
15 x 10 x 3	$\overline{MRAS}$		14	13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																	2	1	0											
	$\overline{MCAS}$																				9	8	7	6	5	4	3	2	1	0	
14 x 10 x 3	$\overline{MRAS}$			13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																	2	1	0											
	$\overline{MCAS}$																				9	8	7	6	5	4	3	2	1	0	
14 x 10 x 2	$\overline{MRAS}$				13	12	11	10	9	8	7	6	5	4	3	2	1	0													
	MBA																		1	0											
	$\overline{MCAS}$																				9	8	7	6	5	4	3	2	1	0	
13 x 10 x 3	$\overline{MRAS}$				12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																	2	1	0											
	$\overline{MCAS}$																				9	8	7	6	5	4	3	2	1	0	
13 x 10 x 2	$\overline{MRAS}$					12	11	10	9	8	7	6	5	4	3	2	1	0													
	MBA																		1	0											
	$\overline{MCAS}$																				9	8	7	6	5	4	3	2	1	0	
13 x 9 x 2	$\overline{MRAS}$						12	11	10	9	8	7	6	5	4	3	2	1	0												
	MBA																			1	0										
	$\overline{MCAS}$																					8	7	6	5	4	3	2	1	0	

Table 9-42. DDR2 Address Multiplexing for 32-bit Data Bus with Interleaving Disabled

Row x 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
15 x 10 x 3	$\overline{MRAS}$			14	13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																		2	1	0											
	$\overline{MCAS}$																					9	8	7	6	5	4	3	2	1	0	
14 x 10 x 3	$\overline{MRAS}$				13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																		2	1	0											
	$\overline{MCAS}$																					9	8	7	6	5	4	3	2	1	0	
14 x 10 x 2	$\overline{MRAS}$					13	12	11	10	9	8	7	6	5	4	3	2	1	0													
	MBA																			1	0											
	$\overline{MCAS}$																					9	8	7	6	5	4	3	2	1	0	

Table 9-42. DDR2 Address Multiplexing for 32-bit Data Bus with Interleaving Disabled (continued)

Row x 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 x 10 x 3	$\overline{\text{MRAS}}$					12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																		2	1	0											
	$\overline{\text{MCAS}}$																					9	8	7	6	5	4	3	2	1	0	
13 x 10 x 2	$\overline{\text{MRAS}}$						12	11	10	9	8	7	6	5	4	3	2	1	0													
	MBA																			1	0											
	$\overline{\text{MCAS}}$																					9	8	7	6	5	4	3	2	1	0	
13 x 9 x 2	$\overline{\text{MRAS}}$							12	11	10	9	8	7	6	5	4	3	2	1	0												
	MBA																				1	0										
	$\overline{\text{MCAS}}$																						8	7	6	5	4	3	2	1	0	

Chip select interleaving is supported for the memory controller, and is programmed in DDR_SDRAM_CFG[BA_INTLV_CTL]. Interleaving is supported between chip selects 0 and 1 or chip selects 2 and 3. In addition, interleaving between all four chip selects can be enabled. When interleaving is enabled, the chip selects being interleaved must use the same size of memory. If two chip selects are interleaved, then 1 extra bit in the address decode is used for the interleaving to determine which chip select to access. If four chip selects are interleaved, then 2 extra bits are required in the address decode.

Table 9-43 illustrates examples of the address decode when interleaving between two chip selects, and Table 9-44 shows examples of the address decode when interleaving between four chip selects.

Table 9-43. Example of Address Multiplexing for 64-bit Data Bus Interleaving Between Two Banks

Row x 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	29–31
14 x 10 x 3	$\overline{\text{MRAS}}$		13	12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL															
	MBA																	2	1	0												
	$\overline{\text{MCAS}}$																				9	8	7	6	5	4	3	2	1	0		
14 x 10 x 2	$\overline{\text{MRAS}}$			13	12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL														
	MBA																		1	0												
	$\overline{\text{MCAS}}$																				9	8	7	6	5	4	3	2	1	0		
13 x 10 x 3	$\overline{\text{MRAS}}$			12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL															
	MBA																	2	1	0												
	$\overline{\text{MCAS}}$																				9	8	7	6	5	4	3	2	1	0		
13 x 10 x 2	$\overline{\text{MRAS}}$				12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL														
	MBA																		1	0												
	$\overline{\text{MCAS}}$																				9	8	7	6	5	4	3	2	1	0		

Table 9-44. Example of Address Multiplexing for 64-bit Data Bus Interleaving Between Four Banks

Row x 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	29–31
14 x 10 x 3	$\overline{\text{MRAS}}$	13	12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL															
	MBA																	2	1	0											
	$\overline{\text{MCAS}}$																				9	8	7	6	5	4	3	2	1	0	
14 x 10 x 2	$\overline{\text{MRAS}}$		13	12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL														
	MBA																		1	0											
	$\overline{\text{MCAS}}$																				9	8	7	6	5	4	3	2	1	0	
13 x 10 x 3	$\overline{\text{MRAS}}$		12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL															
	MBA																	2	1	0											
	$\overline{\text{MCAS}}$																				9	8	7	6	5	4	3	2	1	0	
13 x 10 x 2	$\overline{\text{MRAS}}$			12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL														
	MBA																		1	0											
	$\overline{\text{MCAS}}$																				9	8	7	6	5	4	3	2	1	0	

9.5.3 JEDEC Standard DDR SDRAM Interface Commands

The following section describes the commands and timings the controller uses when operating in DDR2 or DDR modes.

All read or write accesses to DDR SDRAM are performed by the DDR memory controller using JEDEC standard DDR SDRAM interface commands. The SDRAM device samples command and address inputs on rising edges of the memory clock; data is sampled using both the rising and falling edges of DQS. Data read from the DDR SDRAM is also sampled on both edges of DQS.

The following DDR SDRAM interface commands (summarized in Table 9-45) are provided by the DDR controller. All actions for these commands are described from the perspective of the SDRAM device.

- Row activate—Latches row address and initiates memory read of that row. Row data is latched in SDRAM sense amplifiers and must be restored by a precharge command before another row activate occurs.
- Precharge—Restores data from the sense amplifiers to the appropriate row. Also initializes the sense amplifiers in preparation for reading another row in the memory array, (performing another activate command). Precharge must occur after read or write, if the row address changes on the next open page mode access.
- Read—Latches column address and transfers data from the selected sense amplifier to the output buffer as determined by the column address. During each succeeding clock edge, additional data is driven without additional read commands. The amount of data transferred is determined by the burst size which defaults to 4.

- Write—Latches column address and transfers data from the data pins to the selected sense amplifier as determined by the column address. During each succeeding clock edge, additional data is transferred to the sense amplifiers from the data pins without additional write commands. The amount of data transferred is determined by the data masks and the burst size, which is set to four by the DDR memory controller.
- Refresh (similar to $\overline{\text{MCAS}}$ before $\overline{\text{MRAS}}$)—Causes a row to be read in all logical banks (JEDEC SDRAM) as determined by the refresh row address counter. This refresh row address counter is internal to the SDRAM. After being read, the row is automatically rewritten in the memory array. All logical banks must be in a precharged state before executing a refresh. The memory controller also supports posted refreshes, where several refreshes may be executed at once, and the refresh interval may be extended.
- Mode register set (for configuration)—Allows setting of DDR SDRAM options. These options are: $\overline{\text{MCAS}}$ latency, additive latency (for DDR2), write recovery (for DDR2), burst type, and burst length. $\overline{\text{MCAS}}$ latency may be chosen as provided by the preferred SDRAM (some SDRAMs provide $\overline{\text{MCAS}}$ latency {1,2,3}, some provide $\overline{\text{MCAS}}$ latency {1,2,3,4}, etc.). Burst type is always sequential. Although some SDRAMs provide burst lengths of 1, 2, 4, 8, and page size, this memory controller supports a burst length of 4. A burst length of 8 is supported for DDR1 memory only. The mode register set command is performed by the DDR memory controller during system initialization. Parameters such as mode register data, $\overline{\text{MCAS}}$ latency, burst length, and burst type, are set by software in DDR_SDRAM_MODE[SDMODE] and transferred to the SDRAM array by the DDR memory controller after DDR_SDRAM_CFG[MEM_EN] is set. If DDR_SDRAM_CFG[BI] is set to bypass the automatic initialization, then the MODE registers can be configured through software via 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 will place all logical banks in a precharged state.

Table 9-45. DDR SDRAM Command Table

Operation	CKE Prev.	CKE Current	MCS	MRAS	MCAS	MWE	MBA	MA10	MA
Activate	H	H	L	L	H	H	Logical bank select	Row	Row
Precharge select logical bank	H	H	L	L	H	L	Logical bank select	L	X
Precharge all logical banks	H	H	L	L	H	L	X	H	X
Read	H	H	L	H	L	H	Logical bank select	L	Column
Read with auto-precharge	H	H	L	H	L	H	Logical bank select	H	Column
Write	H	H	L	H	L	L	Logical bank select	L	Column
Write with auto-precharge	H	H	L	H	L	L	Logical bank select	H	Column
Mode register set	H	H	L	L	L	L	Opcode	Opcode	Opcode and mode
Auto refresh	H	H	L	L	L	H	X	X	X
Self refresh	H	L	L	L	L	H	X	X	X

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- (or eight-) beat burst read, but ignores the last three (or seven) beats. Single-beat writes are performed by masking the last three (or seven) beats of the four- (or eight-) beat burst using the data mask MDM[0:8]. If ECC is disabled, writes smaller than double words are performed by appropriately activating the data mask. If ECC is enabled, the controller performs a read-modify write.

NOTE

If a second read or write is pending, reads shorter than four beats are not terminated early even if some data is irrelevant.

To accommodate available memory technologies across a wide spectrum of operating frequencies, the DDR memory controller allows the setting of the intervals defined in Table 9-46 with granularity of one memory clock cycle, except for CASLAT, which can be programmed with 1/2 clock granularity.

Table 9-46. DDR SDRAM Interface Timing Intervals

Timing Intervals	Definition
ACTTOACT	The number of clock cycles from a bank-activate command until another bank-activate command within a physical bank. This interval is listed in the AC specifications of the SDRAM as t_{RRD}.
ACTTOPRE	The number of clock cycles from an activate command until a precharge command is allowed. This interval is listed in the AC specifications of the SDRAM as t_{RAS}.
ACTTORW	The number of clock cycles from an activate command until a read or write command is allowed. This interval is listed in the AC specifications of the SDRAM as t_{RCD}.

Table 9-46. DDR SDRAM Interface Timing Intervals (continued)

Timing Intervals	Definition
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 a SDRAM precharge bank command as soon as possible.
CASLAT	Used in conjunction with additive latency to obtain the READ latency. The number of clock cycles between the registration of a READ command by the SDRAM and the availability of the first piece of output data. If a READ command is registered at clock edge n, and the read latency is m clocks, the data is available nominally coincident with clock edge $n + m$.
PRETOACT	The number of clock cycles from a precharge command until an activate or a refresh command is allowed. This interval is listed in the AC specifications of the SDRAM as t_{RP}
REFINT	Refresh interval. Represents the number of memory bus clock cycles between refresh cycles. Depending on DDR_SDRAM_CFG_2[NUM_PR], some number of rows are refreshed in each SDRAM bank during each refresh cycle. The value of REFINT depends on the specific SDRAMs used and the frequency of the interface as t_{RP}
REFREC	The number of clock cycles from the refresh command until an activate command is allowed. This can be calculated by referring to the AC specification of the SDRAM device. The AC specification indicates a maximum refresh to activate interval in nanoseconds.
WR_DATA_DELAY	Provides different options for the timing between a write command and the write data strobe. This allows write data to be sent later than the nominal time to meet the SDRAM timing requirement between the registration of a write command and the reception of a data strobe associated with the write command. The specification dictates that the data strobe may not be received earlier than 75% of a cycle, or later than 125% of a cycle, from the registration of a write command. This parameter is not defined in the SDRAM specification. It is implementation-specific, defined for the DDR memory controller in TIMING_CFG_2.
WRREC	The number of clock cycles from the last beat of a write until a precharge command is allowed. This interval, write recovery time, is listed in the AC specifications of the SDRAM as t_{WR}.
WRTORD	Last write pair to read command. Controls the number of clock cycles from the last write data pair to the subsequent read command to the same bank as t_{WTR}.

The value of the above parameters (in whole clock cycles) must be set by boot code at system start-up (in the TIMING_CFG_0, TIMING_CFG_1, TIMING_CFG_2, and TIMING_CFG_3 registers as described in Section 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 9-35 through Figure 9-37 show DDR SDRAM timing for various types of accesses; see Figure 9-35 for a single-beat read operation, Figure 9-36 for a single-beat write operation, and Figure 9-37 for a burst-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 (for DDR1).

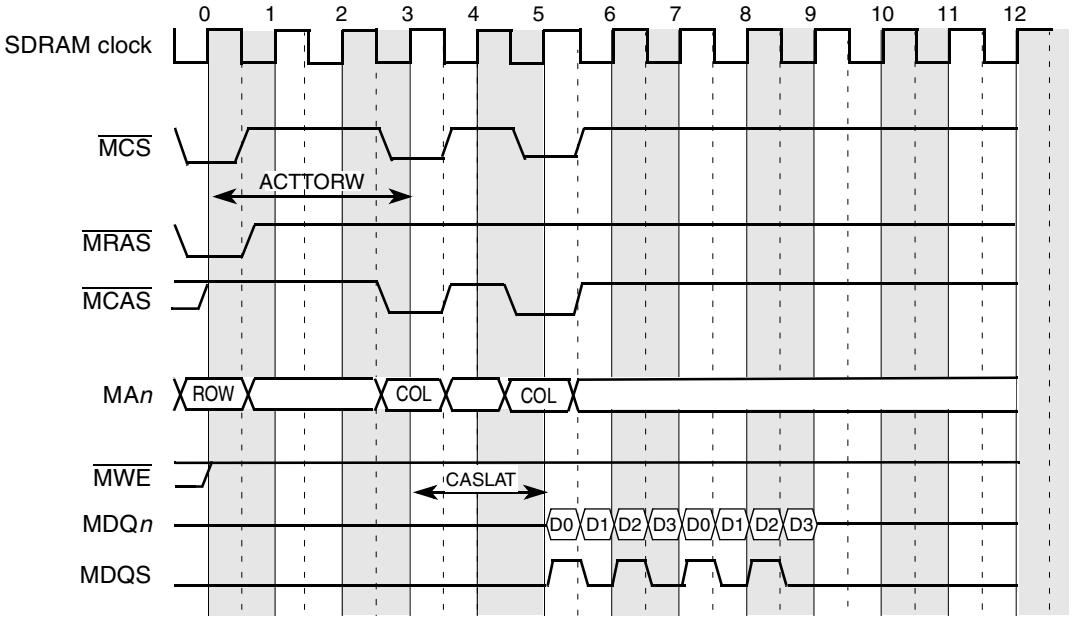


Figure 9-35. DDR SDRAM Burst Read Timing—ACTTORW = 3, MCAS Latency = 2

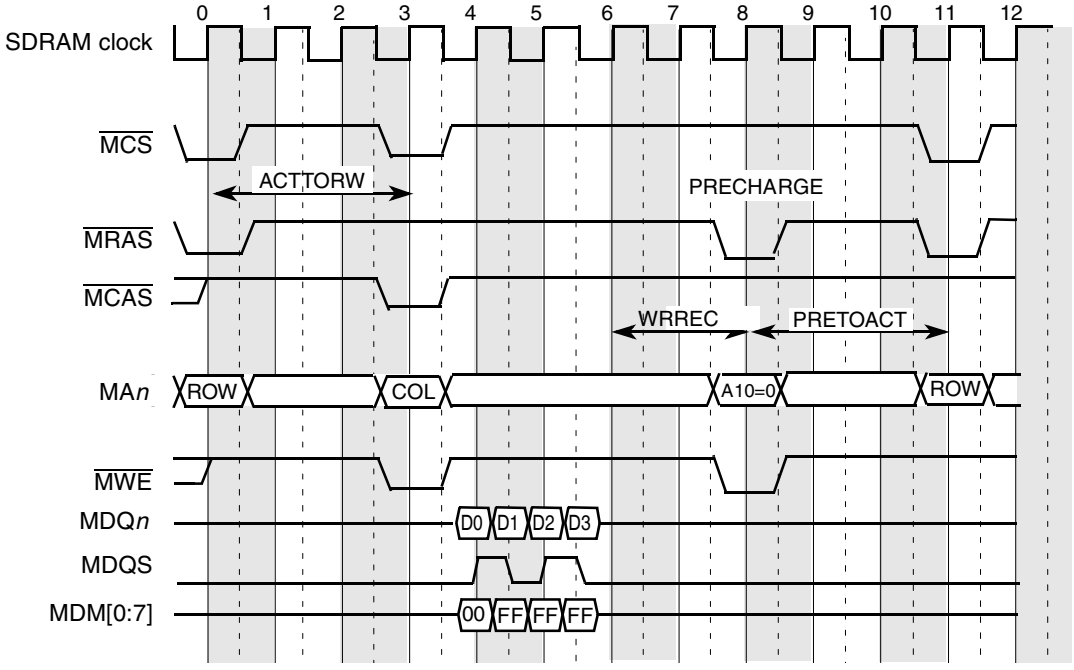


Figure 9-36. DDR SDRAM Single-Beat (Double Word) Write Timing—ACTTORW = 3

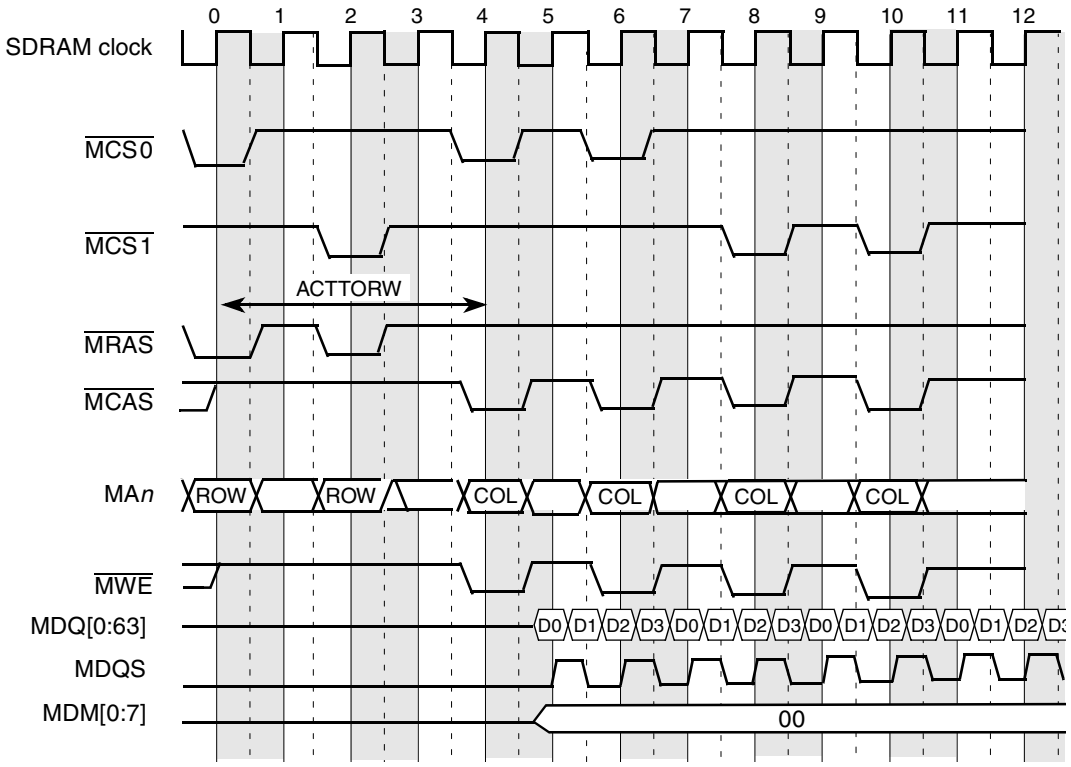


Figure 9-37. DDR SDRAM 4-Beat Burst Write Timing—ACTTORW = 4

9.5.4.1 Clock Distribution

- If running with many devices, zero-delay PLL clock buffers, JEDEC-JESD82 standard, should be used. These buffers were designed for DDR applications.
- A 72 bit x 64 Mbytes DDR bank has 9-byte-wide DDR chips, resulting in 18 DDR chips in a two-bank system. In this case, each MCK/$\overline{\text{MCK}}$ signal pair should drive exactly three devices.
- PCB traces for DDR clock signals should be short, all on the same layer, and of equal length and loading.
- DDR SDRAM manufacturers provide detailed information on PCB layout and termination issues.

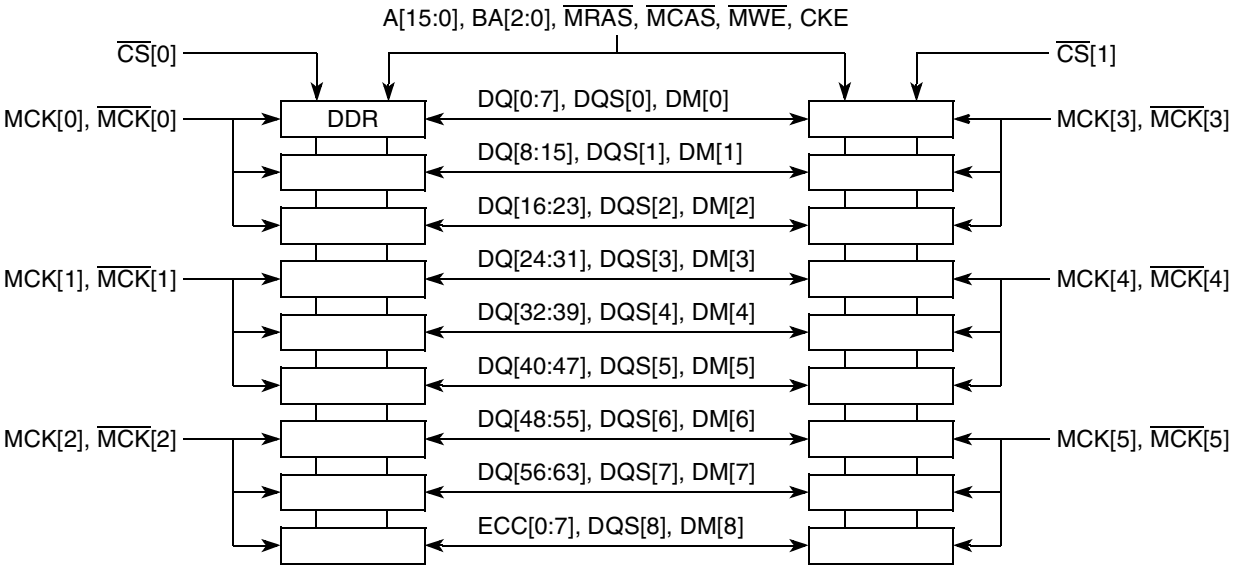


Figure 9-38. DDR SDRAM Clock Distribution Example for x8 DDR SDRAMs

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 9-39 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.

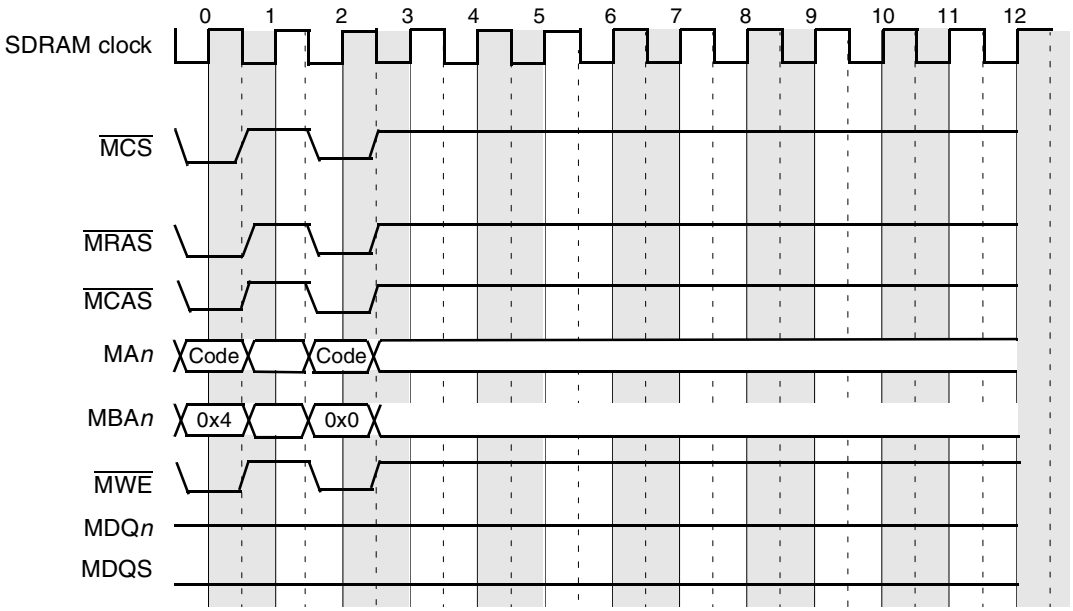


Figure 9-39. DDR SDRAM Mode-Set Command Timing

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 9-40 shows the registered DDR1 SDRAM DIMM single-beat write timing.

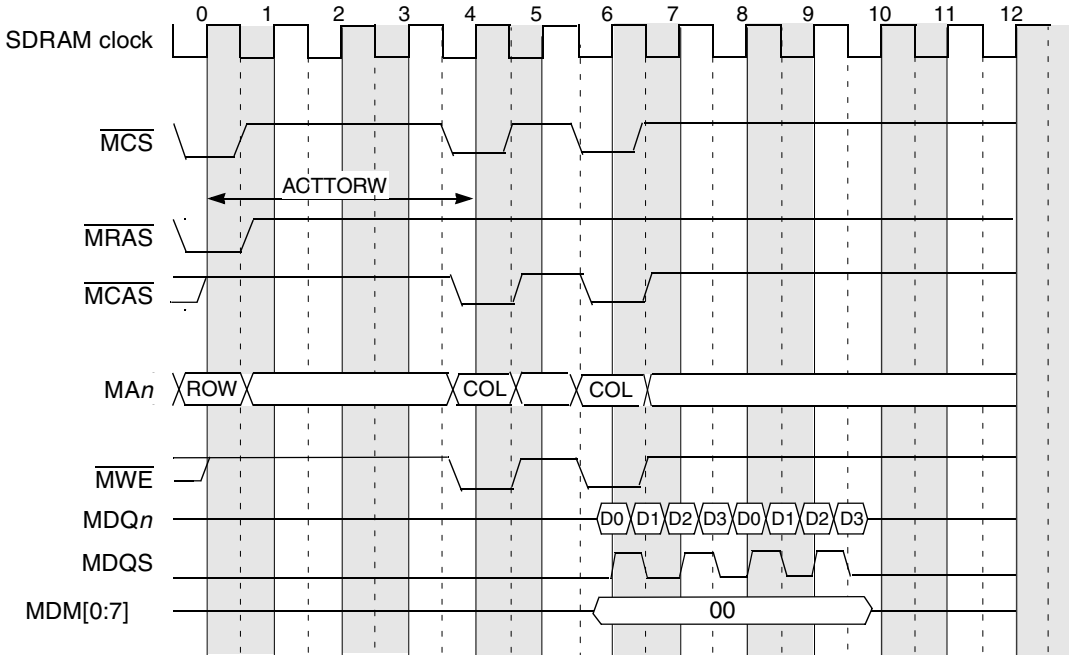


Figure 9-40. Registered DDR1 SDRAM DIMM Burst Write Timing

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. The WR_DATA_DELAY parameter may be used to meet this timing requirement for a variety of system configurations, ranging from a system with one DIMM to a fully populated system with two DIMM. 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 9-41 shows the use of the WR_DATA_DELAY parameter.

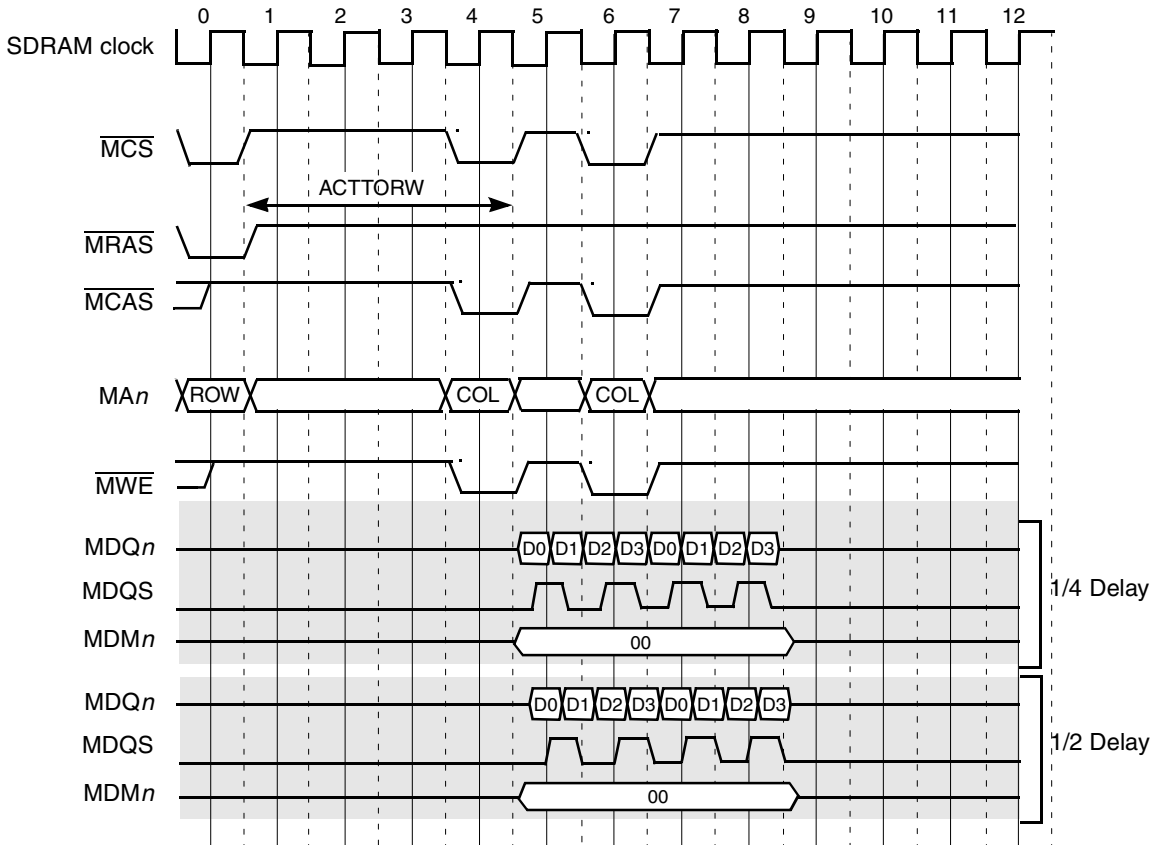


Figure 9-41. Write Timing Adjustments Example for Write Latency = 1

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 four possible banks to reduce the system's instantaneous power requirements. Three sets of auto refresh commands will be issued on consecutive cycles when the memory is fully populated with two DIMMs. The initial PRECHARGE-ALL commands are also staggered in three groups for convenience. It is important to note that when entering self-refresh mode, only one refresh command is issued simultaneously to all physical banks. For this entire refresh sequence, no cycle optimization occurs for the usual case where fewer than four banks are installed. After the refresh sequence completes, any pending memory request is initiated after an inactive period specified by TIMING_CFG_1 [REFREC] and TIMING_CFG_3[EXT_REFREC]. In addition, posted refreshes are supported to allow the refresh interval to be set to a larger value.

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 9-42 (TIMING_CFG_1 [REFREC] = 10 in this example).

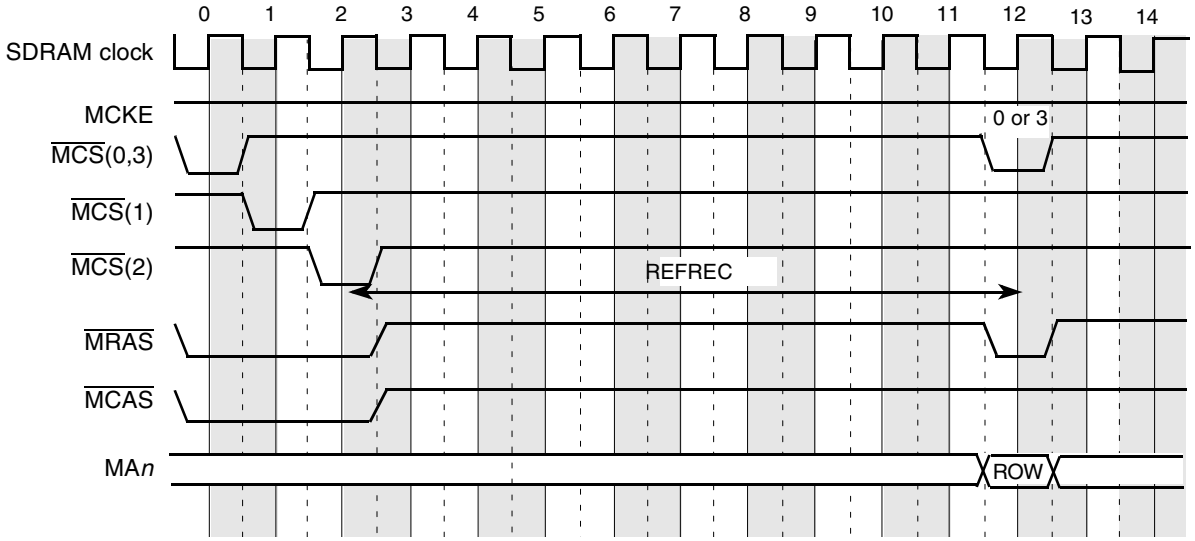


Figure 9-42. DDR SDRAM Bank Staggered Auto Refresh Timing

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 9-47 summarizes the refresh types available in each power-saving mode.

Table 9-47. 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 upon 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 9-43.

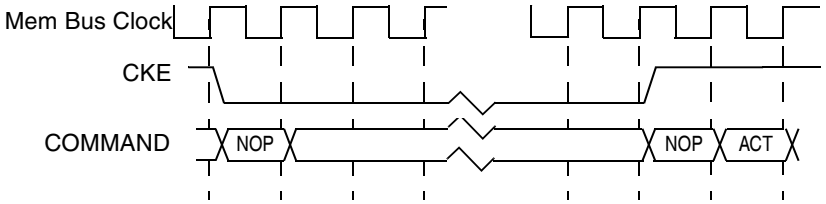


Figure 9-43. DDR SDRAM Power-Down Mode

9.5.8.2.1 Self-Refresh in Sleep Mode

The entry and exit timing for self-refreshing SDRAMs is shown in Figure 9-44 and Figure 9-45.

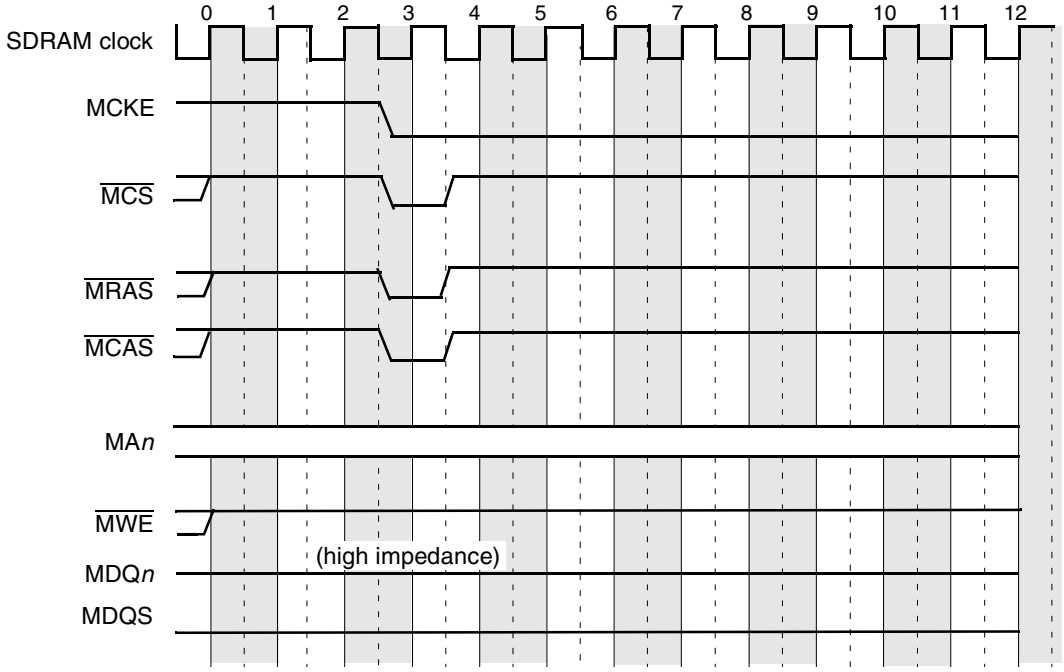


Figure 9-44. DDR SDRAM Self-Refresh Entry Timing

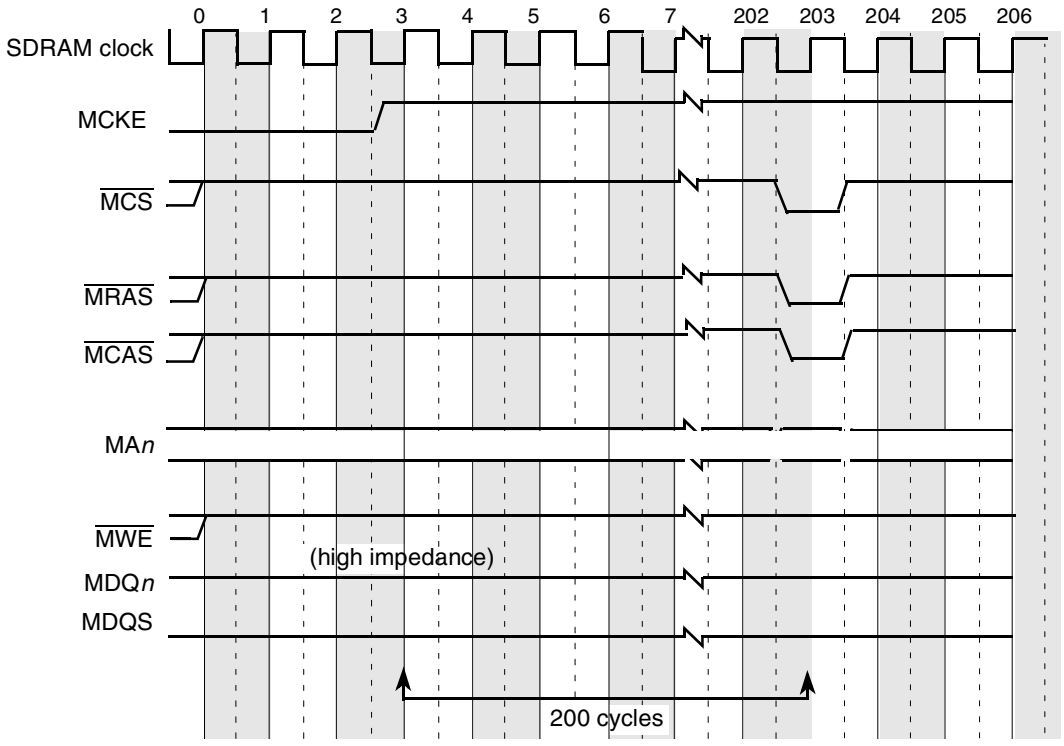


Figure 9-45. DDR SDRAM Self-Refresh Exit Timing

9.5.9 DDR Data Beat Ordering

Transfers to and from memory are always performed in four- or eight-beat bursts (four beats = 32 bytes when a 64-bit bus is used). For transfer sizes other than four or eight beats, the data transfers are still operated as four- or eight-beat bursts. If ECC is enabled and either the access is not doubleword aligned or the size is not a multiple of a doubleword, a full read-modify-write is performed for a write to SDRAM. If ECC is disabled or both the access is doubleword aligned with a size that is a multiple of a doubleword, the data masks (MDM[0:8] (MDM[0:4] for 32-bit bus) can be used to prevent the writing of unwanted data to SDRAM. The DDR memory controller also uses data masks to prevent all unintended full double words from writing to SDRAM. For example, if a write transaction is desired with a size of one double word (8 bytes), then the 2nd, 3rd, and 4th beats of data are not written to DRAM.

Table 9-48 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 9-48. Memory Controller–Data Beat Ordering

Transfer Size	Starting Double-Word Offset	Double-Word Sequence[1] to/from DRAM and Queues
1 double word	0 1 2 3	**0** - 1 - 2 - 3 **1** - 2 - 3 - 0 **2** - 3 - 0 - 1 **3** - 0 - 1 - 2
2 double words	0 1 2	**0 - 1** - 2 - 3 **1 - 2** - 3 - 0 **2 - 3** - 0 - 1
3 double words	0 1	**0 - 1 - 2** - 3 **1 - 2 - 3** - 0
4 double words	0 1 2 3	0 - 1 - 2 - 3 1 - 2 - 3 - 0 2 - 3 - 0 - 1 3 - 0 - 1 - 2

1 All underlined 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 by using MA[10] of the address during the COMMAND phase of the access to enable auto-precharge. Auto-precharge is non-persistent in that it is either enabled or disabled for each individual READ or WRITE command. It can, however, be enabled or disabled separately for each chip select.

When the DDR memory controller operates in open page mode, it retains the currently active SDRAM page by not issuing a precharge command. The page remains opens until one of the following conditions occurs:

- Refresh interval is met.
- The user-programmable DDR_SDRAM_INTERVAL[BSTOPRE] value is exceeded.
- There is a logical bank row collision with another transaction that must be issued.

Page mode can dramatically reduce access latencies for page hits. Depending on the memory system design and timing parameters, using page mode can save two to three clock cycles for subsequent burst accesses that hit in an active page. Also, better performance can be obtained by 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_n_EN].

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 9-49 and Table 9-50.

Table 9-49. DDR SDRAM ECC Syndrome Encoding

Data Bit	Syndrome Bit								Data Bit	Syndrome Bit							
	0	1	2	3	4	5	6	7		0	1	2	3	4	5	6	7
0	•	•						•	32			•	•				•
1	•		•					•	33			•		•			•
2	•			•				•	34	•		•		•			
3	•				•			•	35		•	•		•			
4	•	•				•			36			•	•		•		
5	•		•			•			37			•		•	•		
6	•			•		•			38	•		•		•	•		•
7	•				•	•			39		•	•		•	•		•
8	•	•					•		40			•	•			•	
9	•		•				•		41			•		•		•	
10	•			•			•		42	•		•		•		•	•
11	•				•		•		43		•	•		•		•	•
12	•	•				•	•	•	44			•	•		•	•	•
13	•		•			•	•	•	45			•		•	•	•	•
14	•			•		•	•	•	46	•		•		•	•	•	
15	•				•	•	•	•	47		•	•		•	•	•	
16		•	•					•	48		•				•	•	
17		•		•				•	49			•			•	•	
18		•			•			•	50				•		•	•	
19	•	•			•				51	•					•	•	
20		•	•			•			52		•				•		•
21		•		•		•			53			•			•		•
22		•			•	•			54				•		•		•
23	•	•			•	•		•	55	•					•		•
24		•	•				•		56		•					•	•
25		•		•			•		57			•				•	•
26		•			•		•		58				•			•	•
27	•	•			•		•	•	59	•						•	•
28		•	•			•	•	•	60				•	•		•	
29		•		•		•	•	•	61	•			•	•		•	•
30		•			•	•	•	•	62		•		•	•		•	•
31	•	•			•	•	•		63			•	•	•		•	•

Table 9-50. 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).”

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. For all memory select errors, the DDR memory controller does not issue any transactions onto the pins after the first read has returned data strobes. If the DDR memory controller is not using sample points, then a dummy transaction is issued to DDR SDRAM with the first enabled chip select. In this case, the source port on the pins is forced to 0x1F to show the transaction is not real. Table 9-51 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 9-51. Memory Controller Errors

Category	Error	Descriptions	Action	Detect Register
Notification	Single-bit ECC threshold	The number of ECC errors has reached the threshold specified in the ERR_SBE.	The error is reported via interrupt if enabled.	The error control register only logs read versus write, not full type
Access Error	Multi-bit ECC error	A multi-bit ECC error is detected during a read, or read-modify-write memory operation.		
	Memory select error	Read, or write, address does not fall within the address range of any of the memory banks.		

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 $\overline{\text{MCS}}n$ 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 9-52.

Table 9-52. Memory Interface Configuration Register Initialization Parameters

Name	Description	Parameter		Section/page
CSn_BNDS	Chip select memory bounds	SAn EAn		9.4.1.1/9-10
CSn_CONFIG	Chip select configuration	CS_n_EN AP_n_EN ODT_RD_CFG ODT_WR_CFG	BA_BITS_CS_n ROW_BITS_CS_n COL_BITS_CS_n	9.4.1.2/9-11
TIMING_CFG_3	Extended timing parameters for fields in TIMING_CFG_1	EXT_REFREC		9.4.1.3/9-13
TIMING_CFG_0	Timing configuration	RWT WRT RRT WWT	ACT_PD_EXIT PRE_PD_EXIT ODT_PD_EXIT MRS_CYC	9.4.1.4/9-14
TIMING_CFG_1	Timing configuration	PRETOACT ACTTOPRE ACTTORW CASLAT	REFREC WRREC ACTTOACT WRTORD	9.4.1.5/9-16
TIMING_CFG_2	Timing configuration	ADD_LAT CPO WR_LAT RD_TO_PRE	WR_DATA_DELAY CKE_PLS FOUR_ACT	9.4.1.6/9-18

Table 9-52. Memory Interface Configuration Register Initialization Parameters (continued)

Name	Description	Parameter	Section/page
DDR_SDRAM_CFG	Control configuration	SREN ECC_EN RD_EN SDRAM_TYPE DYN_PWR 32_BE 8_BE NCAP 2T_EN BA_INTLV_CTL x32_EN HSE BI	9.4.1.7/9-20
DDR_SDRAM_CFG_2	Control configuration	SR_IE DLL_RST_DIS DQS_CFG ODT_CFG NUM_PR D_INIT	9.4.1.8/9-22
DDR_SDRAM_MODE	Mode configuration	ESDMODE SDMODE	9.4.1.9/9-24
DDR_SDRAM_MODE_2	Mode configuration	ESDMODE2 ESDMODE3	9.4.1.10/9-24
DDR_SDRAM_INTERVAL	Interval configuration	REFINT BSTOPRE	9.4.1.12/9-27
DDR_DATA_INIT	Data initialization configuration register	INIT_VALUE	9.4.1.13/9-28
DDR_SDRAM_CLK_CNTL	Clock adjust	CLK_ADJUST	9.4.1.14/9-28
DDR_INIT_ADDR	Initialization address	INIT_ADDR	9.4.1.15/9-29

9.6.1 Programming Differences Between Memory Types

Depending upon the memory type used, certain fields must be programmed differently. Table 9-53 illustrates the differences in certain fields for different memory types. Note: This table does not list all fields that must be programmed.

Table 9-53. Programming Differences Between Memory Types

Parameter	Description	Differences		Section/page
AP_*n*_EN	Chip Select *n* Auto Precharge Enable	DDR1	Can be used to place chip select *n* in auto precharge mode	9.4.1.2/9-11
		DDR2	Can be used to place chip select *n* in auto precharge mode	
ODT_RD_CFG	Chip Select ODT Read Configuration	DDR1	Should always be set to 000	9.4.1.2/9-11
		DDR2	Can be enabled to assert ODT if desired. This could be set differently depending upon system topology. However, systems with only 1 chip select will typically not use ODT when issuing reads to the memory.	

Table 9-53. Programming Differences Between Memory Types (continued)

Parameter	Description	Differences		Section/page
ODT_WR_CFG	Chip Select ODT Write Configuration	DDR1	Should always be set to 000	9.4.1.2/9-11
		DDR2	Can be enabled to assert ODT if desired. This could be set differently depending upon system topology. However, ODT will typically be set to assert for the chip select that is getting written to (value would be set to 001).	
ODT_PD_EXIT	ODT Powerdown Exit	DDR1	Should be set to 0000	9.4.1.4/9-14
		DDR2	Should be set according to the DDR2 specifications for the memory used. The JEDEC parameter this applies to is t_{AXPD}.	
PRETOACT	Precharge to Activate Timing	DDR1	Should be set according to the specifications for the memory used (t_{RP})	9.4.1.5/9-16
		DDR2	Should be set according to the specifications for the memory used (t_{RP})	
ACTTOPRE	Activate to Precharge Timing	DDR1	Should be set, along with the Extended Activate to Precharge Timing, according to the specifications for the memory used (t_{RAS})	9.4.1.5/9-16
		DDR2	Should be set, along with the Extended Activate to Precharge Timing, according to the specifications for the memory used (t_{RAS})	
ACTTORW	Activate to Read/Write Timing	DDR1	Should be set according to the specifications for the memory used (t_{RCD})	9.4.1.5/9-16
		DDR2	Should be set according to the specifications for the memory used (t_{RCD})	
CASLAT	CAS Latency	DDR1	Should be set, along with the Extended CAS Latency, to the desired CAS latency	9.4.1.5/9-16
		DDR2	Should be set, along with the Extended CAS Latency, to the desired CAS latency	
REFREC	Refresh Recovery	DDR1	Should be set, along with the Extended Refresh Recovery, to the specifications for the memory used (t_{RFC})	9.4.1.5/9-16
		DDR2	Should be set, along with the Extended Refresh Recovery, to the specifications for the memory used (T_{RFC})	
WRREC	Write Recovery	DDR1	Should be set according to the specifications for the memory used (t_{WR})	9.4.1.5/9-16
		DDR2	Should be set according to the specifications for the memory used (t_{WR})	
ACTTOACT	Activate *A* to Activate *B*	DDR1	Should be set according to the specifications for the memory used (t_{RRD})	9.4.1.5/9-16
		DDR2	Should be set according to the specifications for the memory used (t_{RRD})	

Table 9-53. Programming Differences Between Memory Types (continued)

Parameter	Description	Differences		Section/page
WRTORD	Write to Read Timing	DDR1	Should be set according to the specifications for the memory used (t_{WRD})	9.4.1.5/9-16
		DDR2	Should be set according to the specifications for the memory used (t_{WRD})	
ADD_LAT	Additive Latency	DDR1	Should be set to 000	9.4.1.6/9-18
		DDR2	Should be set to the desired additive latency. This must be set to a value less than TIMING_CFG_1[ACTTORW]	
WR_LAT	Write Latency	DDR1	Should be set to 001	9.4.1.6/9-18
		DDR2	Should be set to CAS latency – 1 cycle. For example, if the CAS latency if 5 cycles, then this field should be set to 100 (4 cycles).	
RD_TO_PRE	Read to Precharge Timing	DDR1	Should be set to 010 if burst length is 4 and 100 if burst length is 8	9.4.1.6/9-18
		DDR2	Should be set according to the specifications for the memory used (t_{RTP}). Time between read and precharge for non-zero value of additive latency (AL) is a minimum of AL + t_{RTP} cycles.	
CKE_PLS	Minimum CKE Pulse Width	DDR1	Can be set to 001	9.4.1.6/9-18
		DDR2	Should be set according to the specifications for the memory used (t_{CKE})	
FOUR_ACT	Four Activate Window	DDR1	Should be set to 0001	9.4.1.6/9-18
		DDR2	Should be set according to the specifications for the memory used (t_{FAW}). Only applies to eight logical banks.	
RD_EN	Registered DIMM Enable	DDR1	If registered DIMMs are used, then this field should be set to 1	9.4.1.7/9-20
		DDR2	If registered DIMMs are used, then this field should be set to 1	
8_BE	8-beat burst enable	DDR1	If a 32-bit bus is used, and 8-beat bursts are desired, then this field should be set to 1	9.4.1.7/9-20
		DDR2	Should be set to 0	
2T_EN	2T Timing Enable	DDR1	In heavily loaded systems, this can be set to 1 to gain extra timing margin on the interface at the cost of address/command bandwidth.	9.4.1.7/9-20
		DDR2	In heavily loaded systems, this can be set to 1 to gain extra timing margin on the interface at the cost of address/command bandwidth.	

Table 9-53. Programming Differences Between Memory Types (continued)

Parameter	Description	Differences		Section/page
DLL_RST_DIS	DLL Reset Disable	DDR1	Should typically be set to 0, unless it is desired to bypass the DLL reset when exiting self refresh.	9.4.1.8/9-22
		DDR2	Should typically be set to 0, unless it is desired to bypass the DLL reset when exiting self refresh.	
DQS_CFG	DQS Configuration	DDR1	Should be set to 00	9.4.1.8/9-22
		DDR2	Can be set to either 00 or 01, depending upon if differential strobes will be used	
ODT_CFG	ODT Configuration	DDR1	Should be set to 00	9.4.1.8/9-22
		DDR2	Can be set for termination at the IOs according to system topology. Typically, if ODT is enabled, then the internal IOs should be set up for termination only during reads to DRAM.	
BSTOPRE	Burst To Precharge Interval	DDR1	Can be set to any value, depending upon the application. Auto precharge can be enabled by setting this field to all 0s.	9.4.1.12/9-27
		DDR2	Can be set to any value, depending upon the application. Auto precharge can be enabled by setting this field to all 0s.	

9.6.2 DDR SDRAM Initialization Sequence

After configuration of all parameters is complete, system software must set DDR_SDRAM_CFG[MEM_EN] to enable the memory interface. Note that 200 µs 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 will conduct an automatic initialization sequence to the memory, which will follow the memory specifications. If the bypass initialization mode is used, then software can initialize the memory through the DDR_SDRAM_MD_CNTL register.

9.6.3 Using Forced Self-Refresh Mode to Implement a Battery-Backed RAM System

This section describes the options offered by this device to support battery-backed main memory.

9.6.3.1 Hardware Based Self-Refresh Scheme

An external voltage sense device can be connected to the MPC8349E through the IRQ1/DDR_SR_REQ signal. The assertion of this signal, when a voltage drop has been identified, is then be sensed by the DDR controller. The DDR controller immediately responds by sending a self refresh command to main memory, telling it to enter self-refresh mode.

The DDR controller's response to the assertion of DDR_SR_REQ is controlled by DDR_SDRAM_CFG_2[SR_IE]; see Section 9.4.1.8, "DDR SDRAM Control Configuration 2 (DDR_SDRAM_CFG_2)," for further information on this bit. Note that if IRQ1/DDR_SR_REQ is defined to be used as DDR_SR_REQ, it precludes any other usage of this pin as IRQ1; thus IRQ1 source should be disabled in the interrupt controller. Section 8.5.11, "System External Interrupt Mask Register (SEMSR)," contains a description of the register used to accomplish this task.

9.6.3.2 Software Based Self-Refresh Scheme

The DDR controller also has a software-programmable bit, DDR_SDRAM_CFG_2[FRC_SR], that immediately puts main memory into self-refresh mode. See Section 9.4.1.8, "DDR SDRAM Control Configuration 2 (DDR_SDRAM_CFG_2)," for a description of this register.

It is expected that a critical interrupt routine triggered by an external voltage sensing device will have time to set this bit.

9.6.3.3 Bypassing Re-initialization During Battery-Backed Operation

The DDR controller offers an initialization bypass feature (DDR_SDRAM_CFG[BI]), which system designers may use to prevent re-initialization of main memory during system power-on following an abnormal shutdown. See Section 9.4.1.7, "DDR SDRAM Control Configuration (DDR_SDRAM_CFG)," for information on this bit and Section 9.4.1.15, "DDR Initialization Address (DDR_INIT_ADDR)," for a discussion of avoiding possible ECC errors in this mode.

Note that the DDR controller will automatically wait 200 DRAM cycles before issuing any command after the assertion of MCKE[0:1] when this mode is used.

Chapter 10
Local Bus Controller

This chapter describes the local bus controller (LBC) block. It describes the external signals and the memory-mapped registers and contains a functional description of the general-purpose chip-select machine (GPCM), synchronous DRAM (SDRAM) machine, and user-programmable machines (UPMs) of the LBC. Finally, it includes initialization and applications information sections with many specific examples of its use.

10.1 Introduction

Figure 10-1 is a functional block diagram of the LBC, which supports three interfaces: GPCM, UPM, and an SDRAM controller.

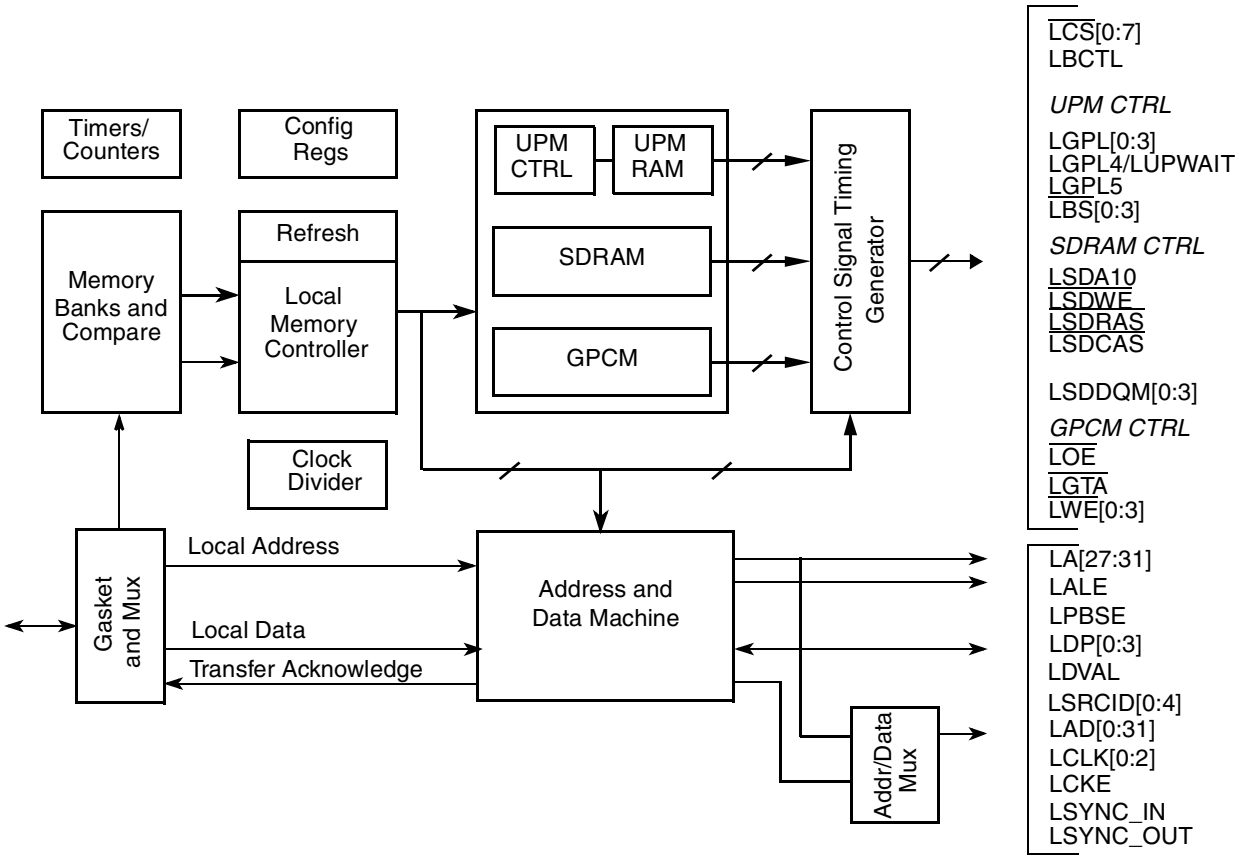


Figure 10-1. Local Bus Controller Block Diagram

The main component of the LBC is its memory controller, which provides a seamless interface to many types of memory devices and peripherals. The memory controller is responsible for controlling eight memory banks shared by a high performance SDRAM machine, a GPCM, and up to three UPMs. As such, it supports a minimal glue logic interface to SDRAM, SRAM, EPROM, flash EPROM, burstable RAM, regular DRAM devices, extended data output DRAM devices, and other peripherals. The external address latch signal (LALE) allows multiplexing of addresses with data signals to reduce the device signal count.

The LBC 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.

10.1.1 Features

The LBC main features are as follows:

- Memory controller with eight memory banks
  - 32-bit address decoding with mask
  - Variable memory block sizes (32 Kbytes to 2 Gbytes)
  - Selection of control signal generation on a per-bank basis
  - Data buffer controls activated on a per-bank basis
  - Automatic segmentation of large transactions
  - Odd/even parity checking including read-modify-write (RMW) parity for single accesses
  - Write-protection capability
  - Parity byte-select
- SDRAM machine
  - Provides the control functions and signals for glueless connection to JEDEC-compliant SDRAM devices
  - Supports up to 4 concurrent open pages per device
  - Supports SDRAM port size of 32-bit, 16-bit and 8-bit
  - Supports external address and/or command lines buffering
- General-purpose chip-select machine (GPCM)
  - Compatible with SRAM, EPROM, FEPROM, and peripherals
  - Global (boot) chip-select available at system reset
  - Boot chip-select support for 8-, 16-, or 32-bit devices
  - Minimum three-clock access to external devices
  - Four byte-write-enable signals ($\overline{\text{LWE}}$[0:3])
  - Output enable signal ($\overline{\text{LOE}}$)
  - External access termination signal ($\overline{\text{LGTA}}$)

- Three user-programmable machines (UPMs)
  - Programmable-array-based machine controls external signal timing with a granularity of up to one quarter of an external bus clock period
  - User-specified control-signal patterns run when an internal master requests a single-beat or burst read or write access.
  - UPM refresh timer runs a user-specified control signal pattern to support refresh
  - User-specified control-signal patterns can be initiated by software
  - Each UPM can be defined to support DRAM devices with depths of 64, 128, 256, and 512 Kbytes, and 1, 2, 4, 8, 16, 32, 64, 128, and 256 Mbytes
  - Support for 8-, 16-, 32-bit devices
  - Page mode support for successive transfers within a burst
  - Internal address multiplexing supporting 64-, 128-, 256-, and 512-Kbyte, and 1-, 2-, 4-, 8-, 16-, 32-, 64-, 128-, and 256-Mbyte page banks
- Optional monitoring of transfers between local bus internal masters and local bus slaves (local bus error reporting)
- Support for delay-locked loop (DLL) with software-configurable bypass for low frequency bus clocks

10.1.2 Modes of Operation

The LBC provides one GPCM, one SDRAM machine, and three UPMs for the local bus, with no restriction on how many of the eight banks (chip selects) can be programmed to operate with any given machine. When a memory transaction is dispatched to the LBC, the memory address is compared with the address information of each bank (chip select). The corresponding machine assigned to that bank (GPCM, SDRAM, or UPM) then takes ownership of the external signals that control the access and maintains control until the transaction ends. Thus, with the LBC in GPCM, SDRAM, or UPM mode, only one of the eight chip selects is active at any time for the duration of the transaction. See Section 10.4, “Functional Description.”

10.1.2.1 LBC Bus Clock and Clock Ratios

The LBC supports ratios of 2, 4, and 8 between the faster internal (system) clock and slower external bus clock (LCLK[0:2]). This ratio is software programmable through the clock ratio register (LCRR[CLKDIV]). This ratio has no functional effect on operation in SDRAM mode but it affects the resolution of signal timing shifts in GPCM mode and the interpretation of UPM array words in UPM mode. The bus clock is driven identically onto signals, LCLK[0:2] to allow the clock load to be shared equally across a pair of signal nets, thereby enhancing the edge rates of the bus clock.

10.1.2.2 Source ID Debug Mode

The LBC provides the ID of a transaction source on external device signals. When those signals 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 LBC external signals. The reserved value of 0x1F, which indicates invalid address or data, appears on the source ID signals at all other times. The combination of a valid

source ID (any value except 0x1F) and the value of external address latch enable (LALE) and data valid (LDVAL) facilitate capturing useful debug data as follows:

- If a valid source ID is detected on LSRCID[0:4] and LALE is asserted, a valid full 32-bit address may be latched from LAD[0:31]. Note that in SDRAM mode the address vector contains the full address as {row, bank, column, lsb's} where row corresponds to the same row address for the given column address and lsb's are the unconnected lsb's of the address for a given port size.
- If a valid source ID is detected on LSRCID[0:4] and LDVAL is asserted, valid data may be latched from LAD[0:31].

The LSRCID[0:4] and LDVAL signals are multiplexed with other functions sharing the same external signals. Refer to chapter 3, external signals description and to chapter 5, system configuration to learn how to enable the LSRCID/LDVAL signals.

10.2 External Signal Descriptions

Table 10-1 contains a list of external signals related to the LBC 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 MPC8349EA, see Section 4.2.2, "Power-On Reset Flow." Note that during assertion of $\overline{\text{HRESET}}$, the DLL is initially unlocked, so the LCLK and LSYNC_OUT values are likely to be unstable/jittery for several microseconds; after the DLL locks, stable clock signals are driven on these signals.

Table 10-1. Signal Properties—Summary

Name	Alternate Functions	Mode	Descriptions	No. of Signals	I/O	Reset State (Outputs)
LALE	—	—	External address latch enable	1	O	Reset_cfg
$\overline{\text{LCS0}}$	—	—	Chip select 0	1	O	Reset_cfg
$\overline{\text{LCS}}$[1:7]	—	—	Chip selects [1–7]	7	O	All high
$\overline{\text{LWE}}$[0:3]/ LSDDQM/ $\overline{\text{LBS}}$[0:3]	$\overline{\text{LWE}}$[0:3]	GPCM	Write enable	4	O	Reset_cfg
	SDDQM	SDRAM	Byte lane data mask			
	$\overline{\text{LBS}}$[0:3]	UPM	Byte (lane) select			
LGPL0 LSDA10	LGPL0	UPM	General purpose line 0	1	O	Reset_cfg
	LSDA10	SDRAM	Row address bit/command bit			
LGPL1 $\overline{\text{LSDWE}}$	LGPL1	UPM	General purpose line 1	1	O	Reset_cfg
	$\overline{\text{LSDWE}}$	SDRAM	Write enable			
$\overline{\text{LOE}}$/ $\overline{\text{LSDRAS}}$/ LGPL2	$\overline{\text{LOE}}$	GPCM	Output enable	1	O	
	$\overline{\text{LSDRAS}}$	SDRAM	Row address strobe			
	LGPL2	UPM	General purpose line 2			
LGPL3 $\overline{\text{LSDCAS}}$	LGPL3	UPM	General purpose line 3	1	O	Reset_cfg
	$\overline{\text{LSDCAS}}$	SDRAM	Column address strobe			

Table 10-1. Signal Properties—Summary (continued)

Name	Alternate Functions	Mode	Descriptions	No. of Signals	I/O	Reset State (Outputs)
$\overline{\text{LGTA}}$/ LGPL4/ LUPWAIT/ LPBSE	$\overline{\text{LGTA}}$	GPCM	Transaction termination	1	I	High-Z
	LGPL4	UPM	General purpose line 4		O	
	LUPWAIT	UPM	External device wait		I	
	LPBSE	—	Local bus parity byte select		O	
LGPL5	—	UPM	General purpose line 5	1	O	Reset_cfg
LBCTL	—	—	Data buffer control	1	O	
LA[27:31]	—	—	Local bus non-multiplexed address lsb's	5	O	
LAD[0:31]	—	—	Multiplexed address/data bus	32	I/O	
LDP	—	—	Local bus data parity	4	I/O	High-Z
LCKE	—	—	Local bus clock enable	1	O	High
LCLK[0:2]	—	—	Local bus clocks	3	O	Driven
LSYNC_IN	—	—	DLL synchronize input	1	I	—
LSYNC_OUT	—	—	DLL synchronize output	1	O	Driven
LDVAL	—	LBC debug	Local bus data valid	1	O	Not connected to external signals
LSRCID[0:4]	—	LBC debug	Local bus source ID	5	O	Not connected to external signals

Table 10-2 shows the detailed external signal descriptions for the LBC.

Table 10-2. Local Bus Controller Detailed Signal Descriptions

Signal	I/O	Description	
LALE	O	External address latch enable. The local bus memory controller provides control for an external address latch, which allows address and data to be multiplexed on the device signals. See Section 10.4.1.2, "External Address Latch Enable Signal (LALE)."	
		State Meaning	Asserted/Negated—LALE is asserted with the address at the beginning of each memory controller transaction. The number of cycles for which it is asserted is governed by the ORn[EAD] and LCRR[EADC] fields. The exact timing of the negation of LALE is controlled by the RCWH[LALE] field. Note that no other control signals are asserted during the assertion of LALE.
$\overline{\text{LCS}}$[0:7]	O	Chip selects. Eight chip selects are provided which are mutually exclusive.	
		State Meaning	Asserted/Negated—Used to enable specific memory devices or peripherals connected to the LBC. $\overline{\text{LCS}}$[0:7] are provided on a per-bank basis with $\overline{\text{LCS0}}$ corresponding to the chip select for memory bank 0, which has the memory type and attributes defined by BR0 and OR0.

Table 10-2. Local Bus Controller Detailed Signal Descriptions (continued)

Signal	I/O	Description	
$\overline{\text{LWE}}$[0:3]/ LSDDQM[0:3]/ $\overline{\text{LBS}}$[0:3]	O	GPCM write enable/SDRAM data mask/UPM byte select. These signals select or validate each byte lane of the data bus. For banks with port sizes of 32 bits (as set by BR*n*[PS]), all four signals are defined. For a 16-bit port size, only bits 0–1 are defined; and for an 8-bit port size, bit 0 is the only defined signal. The least-significant address bits of each access also determine which byte lanes are considered valid for a given data transfer.	
		State Meaning	Asserted/Negated—For GPCM operation, $\overline{\text{LWE}}$[0:3] assert for each byte lane enabled for writing. For SDRAM operation, LSDDQM[0:3] function as the DQM or data mask signals provided by JEDEC-compliant SDRAM devices, with one DQM provided per byte lane. LSDDQM[0:3] are driven high when the LBC wishes to mask a write or disable read data output from the SDRAM. $\overline{\text{LBS}}$[0:3] are programmable byte-select signals in UPM mode. See Section 10.4.4.4, "UPM RAM Array," for programming details about $\overline{\text{LBS}}$[0:3].
		Timing	Assertion/Negation—See Section 10.4.2, "General-Purpose Chip-Select Machine (GPCM)," for details regarding the timing of $\overline{\text{LWE}}$[0:3].
LSDA10/ LGPL0	O	SDRAM A10/General purpose line 0	
		State Meaning	Asserted/Negated—For SDRAM accesses, represents address bit 10. When the row address is driven, it drives the value of address bit 10. When the column address is driven, it forms part of the SDRAM command. One of six general purpose signals when in UPM mode; it drives a value programmed in the UPM array.
$\overline{\text{LSDWE}}$/ LGPL1	O	SDRAM write enable/General-purpose line 1	
		State Meaning	Asserted/Negated—Should be connected to the SDRAM device WE input. Acts as the SDRAM write enable when accessing SDRAM. One of six general purpose signals when in UPM mode, and drives a value programmed in the UPM array.
$\overline{\text{LOE}}$/ $\overline{\text{LSDRAS}}$/ LGPL2	O	GPCM output enable/SDRAM RAS/General-purpose line 2	
		State Meaning	Asserted/Negated—Controls the output buffer of memory when accessing memory/devices in GPCM mode. For SDRAM accesses, it is the row address strobe (RAS). One of six general purpose lines when in UPM mode; it drives a value programmed in the UPM array.
$\overline{\text{LSDCAS}}$/ LGPL3	O	SDRAM CAS/General-purpose line 3	
		State Meaning	Asserted/Negated—In SDRAM mode, drives the column address strobe (CAS). This signal is one of six general purpose signals when in UPM mode, and drives a value programmed in the UPM array.

Table 10-2. Local Bus Controller Detailed Signal Descriptions (continued)

Signal	I/O	Description	
$\overline{\text{LGTA}}$/ LGPL4/ LUPWAIT/ LPBSE	I/O	GPCM transfer acknowledge/General-purpose line 4/UPM wait/parity byte select	
		State Meaning	Asserted/Negated—Input in GPCM mode 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. When configured as LPBSE, it disables any use in GPCM or UPM modes. Because systems that use read-modify-write parity require an additional memory device, they must generate a byte-select like a normal data device. ANDing $\overline{\text{LBS}}$[0:3] through external logic to achieve the logical function of this byte-select adds a delay to the byte-select path that can affect memory access timing. The LBC provides this optional byte-select signal that is an internal AND of the four (active low) byte selects, allowing glueless, faster connection to n -parity devices. **Note:** The $\overline{\text{LGTA}}$/LUPWAIT signal can be sampled as asserted (active-low) during LGPL4/LPBSE is brought low by the UPM. For a subsequent GPCM transaction, it functions as $\overline{\text{LGTA}}$/LUPWAIT. As a result, GPCM transactions may be terminated prematurely before $\overline{\text{LGTA}}$/LUPWAIT has drifted to a logical one. **Work Around:** One way to resolve this issue is to ensure that $\overline{\text{LGTA}}$/LGPL4 is pulled-up to 3.3 V by an external 1-KΩ register. This will ensure that $\overline{\text{LGTA}}$ is sampled high (not asserted) by the time any subsequent GPCM transactions are initiated by the local bus memory controller. If this signal is used purely as an input ($\overline{\text{LGTA}}$/LUPWAIT), a weaker (10-KΩ) pull-up register may be substituted; alternatively, if this signal is used as LPBSE, no pull-up is required, as $\overline{\text{LGTA}}$/LUPWAIT are disabled. A software work around to this errata is to program UPM. Therefore, it can pre-drive LGPL4 high prior to switching to input mode. A weak (10-KΩ or greater) pull-up is still required to maintain a stable level on $\overline{\text{LGTA}}$ for GPCM purposes.
LGPL5	O	General-purpose line 5	
		State Meaning	Asserted/Negated—One of six general purpose signals when in UPM mode, and drives a value programmed in the UPM array.
LBCTL	O	Data buffer control. The memory controller activates LBCTL for the local bus when a GPCM- or UPM-controlled bank is accessed. Access to an SDRAM machine-controlled bank does not activate the buffer control. Buffer control is disabled by setting OR*n*[BCTLD].	
		State Meaning	Asserted/Negated—Normally functions as a write/$\overline{\text{read}}$ control for a bus transceiver connected to the LAD lines. Note that an external data buffer must not drive the LAD lines in conflict with the LBC when LBCTL is high, because LBCTL remains high after reset and during address phases.
LA[27:31]	O	Local bus nonmultiplexed address lsb's. All bits driven on LA[27:31] are defined for 8-bit port sizes. For 32-bit port sizes, LA[30:31] are don't cares; for 16-bit port sizes LA31 is a don't care.	
		State Meaning	Asserted/Negated—Although the LBC shares an address and data bus, up to five lsbs of the RAM address always appear on the dedicated address signals, LA[27:31]. These may be used, unlatched, in place of LAD[27:31] to connect the five lsbs of the address for address phases. For some RAM devices, such as fast-page DRAM, LA[27:31] serve as the column address offset during a burst access.

Table 10-2. Local Bus Controller Detailed Signal Descriptions (continued)

Signal	I/O	Description	
LAD[0:31]	I/O	Multiplexed address/data bus. For configuration of a port size in BR*n*[PS] as 32 bits, all of LAD[0:31] must be connected to the external RAM data bus, with LAD[0:7] occupying the most significant byte lane (at address offset 0). For a port size of 16 bits, LAD[0:7] connect to the most significant byte lane (at address offset 0), while LAD[8:15] connect to the least significant byte lane (at address offset 1); LAD[16:31] are unused for 16-bit port sizes. For a port size of 8 bits, only LAD[0:7] are connected to the external RAM.	
		State Meaning	Asserted/Negated—LAD[0:31] is the shared 32-bit address/data bus through which external RAM devices transfer data and receive addresses
		Timing	Assertion/Negation—During assertion of LALE, LAD[0:31] are driven with the RAM address for the access to follow. External logic should propagate the address on LAD[0:31] while LALE is asserted, and latch the address upon negation of LALE. After LALE is negated, LAD[0:31] are either driven by write data or are made high-impedance by the LBC in order to sample read data driven by an external device. Following the last data transfer of a write access, LAD[0:31] are again taken into a high-impedance state.
LDP[0:3]	I/O	Local bus data parity. Drives and receives the data parity corresponding with the data phases on LAD[0:31].	
		State Meaning	Asserted/Negated—During write accesses, a parity bit is generated for each 8 bits of LAD[0:31], such that LDP0 is even/odd parity for LAD[0:7], while LDP[3] is even/odd parity for LAD[24:31]. Unused byte lanes for port sizes less than 32 bits have undefined parity.
		Timing	Assertion/Negation—Drive and receive the data parity corresponding with the data phases on LAD[0:31]. For read accesses, the parity bits for each byte lane are sampled on LDP[0:3] with the same timing that read data is sampled on LAD[0:31]. LDP[0:3] change impedance in concert with LAD[0:31].
LCKE	O	Local bus clock enable	
		State Meaning	Asserted/Negated—Bus clock enable signal (CKE) for JEDEC-standard SDRAM devices. Asserted during normal SDRAM operation.
LCLK[0:2]	O	Local bus clocks. Enabled by the DLLCK register. See Section 22.4.3, “DLL Clock Register (DLLCK),” for details.	
		State Meaning	Asserted/Negated—LCLK[0:2] drive an identical bus clock signal for distributed loads. If the LBC DLL is enabled (see LCRR[DBYP], Figure 10-19 on page 10-30), the bus clock phase is shifted earlier than transitions on other LBC signals (such as LAD[0:310:15] and $\overline{\text{LCS}n}$) by a time delay matching the delay of the DLL timing loop set up between LSYNC_OUT and LSYNC_IN. DLLLCK
LSYNC_OUT	O	DLL synchronization out.	
		State Meaning	Asserted/Negated—A replica of the bus clock, appearing on LSYNC_OUT, should be propagated through a passive timing loop and returned to LSYNC_IN for achieving correct DLL lock.
		Timing	Assertion/Negation—The time delay of the timing loop should be such that it compensates for the round-trip flight time of LCLK[0:2] and clocked drivers in the system. No load other than a timing loop should be placed on LSYNC_OUT.
LSYNC_IN	I	DLL synchronization in.	
		State Meaning	Asserted/Negated—See description of LSYNC_OUT.

Table 10-2. Local Bus Controller Detailed Signal Descriptions (continued)

Signal	I/O	Description	
LDVAL	O	Local bus data valid (LBC debug mode only)	
		State Meaning	Asserted/Negated—For a read, LDVAL asserts for one bus cycle in the cycle immediately preceding the sampling of read data on LAD[0:31]. For a write, LDVAL asserts for one bus cycle during the final cycle for which the current write data on LAD[0:31] is valid. During burst transfers, LDVAL asserts for each data beat.
		Timing	Assertion/Negation—Valid only while the LBC is in system debug mode. In debug mode, LDVAL asserts when the LBC generates a data transfer acknowledge.
LSRCID[0:4]	O	Local bus source ID (LBC debug mode only). In debug mode, all LSRCID[0:4] signals are driven high unless LSRCID[0:4] is driving a debug source ID for identifying the internal system device controlling the LBC.	
		State Meaning	Asserted/Negated—Remain high until the last bus cycle of the assertion of LALE, in which case the source ID of the address is indicated, or until LDVAL is asserted, in which case the source ID relating to the data transfer is indicated. In case of address debug, LSRCID[0:4] is valid only when the address on LAD[0:31] consists of all physical address bits—with optional padding—for reconstructing the system address presented to the LBC. For example, LSRCID[0:4] is valid only during CAS phases of SDRAM accesses, because the column, bank select, and (normally unused) row address bits are all present on LAD[0:31] during a CAS cycle.

10.3 Memory Map/Register Definition

Table 10-3 shows the memory-mapped registers of the LBC. Undefined 4-byte address spaces within offset 0x000–0xFFF are reserved.

Table 10-3. Local Bus Controller Memory Map

Address Offset	Use	Access	Reset	Section/Page
0x0_5000	BR0—Base register 0	R/W	0x0000_RR01[1]	10.3.1.1/10-11
0x0_5008	BR1—Base register 1		0x0000_0000	
0x0_5010	BR2—Base register 2			
0x0_5018	BR3—Base register 3			
0x0_5020	BR4—Base register 4			
0x0_5028	BR5—Base register 5			
0x0_5030	BR6—Base register 6			
0x0_5038	BR7—Base register 7			

Table 10-3. Local Bus Controller Memory Map (continued)

Address Offset	Use	Access	Reset	Section/Page
0x0_5004	OR0—Option register 0	R/W	0x0000_0FF7	10.3.1.2/10-12
0x0_500C	OR1—Option register 1		0x0000_0000	
0x0_5014	OR2—Option register 2			
0x0_501C	OR3—Option register 3			
0x0_5024	OR4—Option register 4			
0x0_502C	OR5—Option register 5			
0x0_5034	OR6—Option register 6			
0x0_503C	OR7—Option register 7			
0x0_5068	MAR—UPM address register	R/W	0x0000_0000	10.3.1.3/10-18
0x0_5070	MAMR—UPMA mode register	R/W	0x0000_0000	10.3.1.4/10-19
0x0_5074	MBMR—UPMB mode register	R/W	0x0000_0000	10.3.1.4/10-19
0x0_5078	MCMR—UPMC mode register	R/W	0x0000_0000	10.3.1.4/10-19
0x0_5084	MRTPR—Memory refresh timer prescaler register	R/W	0x0000_0000	10.3.1.5/10-21
0x0_5088	MDR—UPM data register	R/W	0x0000_0000	10.3.1.6/10-22
0x0_5094	LSDMR—SDRAM mode register	R/W	0x0000_0000	10.3.1.7/10-22
0x0_50A0	LURT—UPM refresh timer	R/W	0x0000_0000	10.3.1.8/10-24
0x0_50A4	LSRT—SDRAM refresh timer	R/W	0x0000_0000	10.3.1.9/10-25
0x0_50B0	LTESR—Transfer error status register	Read/bit-reset	0x0000_0000	10.3.1.10/10-26
0x0_50B4	LTEDR—Transfer error disable register	R/W	0x0000_0000	10.3.1.11/10-27
0x0_50B8	LTEIR—Transfer error interrupt register	R/W	0x0000_0000	10.3.1.12/10-27
0x0_50BC	LTEATR—Transfer error attributes register	R/W	0x0000_0000	10.3.1.13/10-28
0x0_50C0	LTEAR—Transfer error address register	R/W	0x0000_0000	10.3.1.14/10-29
0x0_50D0	LBCR—Configuration register	R/W	0x0000_0000	10.3.1.15/10-29
0x0_50D4	LCRR—Clock ratio register	R/W	0x8000_0008	10.3.1.16/10-30

[1] Port size for BR0 is configured from the value of RCWH[ROMLOC], which is loaded during reset, hence ‘RR’ is either 0x08, 0x10, or 0x18.

10.3.1 Register Descriptions

This section provides a detailed description of the LBC configuration, status, and control registers with detailed bit and field descriptions.

Address offsets in the LBC 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–BR7)

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, BR0[V] is set, BR1[V]–BR7[V] are cleared, and the value of BR0[PS] reflects the initial port size configured by the boot ROM location field of the reset configuration word.

Offset 0x0_5000 (BR0) 0x0_5020 (BR4) Access: read/write
0x0_5008 (BR1) 0x0_5028 (BR5)
0x0_5010 (BR2) 0x0_5030 (BR6)
0x0_5018 (BR3) 0x0_5038 (BR7)

	0															16	17	18	19	20	21	22	23	24		26	27	28		30	31
R W	BA															—			PS	DECC			WP	MSEL			—	—			V
Reset[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	P	S	0	0	0	0	0	0	0	0	0	0	0/1

[1] BR0 has its valid bit set during reset. Thus bank 0 is valid with the port size (PS) configured from RCWH[ROMLOC] as loaded during reset. All other base registers have all bits cleared to zero during reset.

Figure 10-2. Base Registers (BR*n*)

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 BR0, PS is configured from the boot ROM location 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 10 16-bit 11 32-bit
21–22	DECC	Specifies the method for data error checking. 00 Data error checking disabled, but normal parity generation 01 Normal parity generation and checking 10 Read-modify-write parity generation and normal parity checking (32-bit port size only) 11 Reserved

Table 10-4. BR*n* Field Descriptions (continued)

Bits	Name	Description
23	WP	Write protect. 0 Read and write accesses are allowed. 1 Only read accesses are allowed. The memory controller does not assert $\overline{\text{LCS}n}$ on write cycles to this memory bank. LTESR[WP] is set (if WP is set) if a write to this memory bank is attempted, and a local bus error interrupt is generated (if enabled), terminating the cycle.
24–26	MSEL	Machine select. Specifies the machine to use for handling memory operations. 000 GPCM (reset value) 001 Reserved 010 Reserved 011 SDRAM 100 UPMA 101 UPMB 110 UPMC 111 Reserved
27	—	Reserved
28–29	—	Reserved
30	—	Reserved
31	V	Valid bit. Indicates that the contents of the BR*n* and OR*n* pair are valid. $\overline{\text{LCS}n}$ does not assert unless V is set (an access to a region that has no valid bit set may cause a bus time-out). After a system reset, only BR0[V] is set. 0 This bank is invalid. 1 This bank is valid.

10.3.1.2 Option Registers (OR0–OR7)

The OR*n* registers define the sizes of memory banks and access attributes. The OR*n* attribute bits support the following three modes of operation as defined by BR*n*[MSEL].

- GPCM mode (see Section 10.3.1.2.2, “Option Registers (ORn)—GPCM Mode.”)
- UPM mode (see Section 10.3.1.2.3, “Option Registers (ORn)—UPM Mode.”)
- SDRAM mode (see Section 10.3.1.2.4, “Option Registers (ORn)—SDRAM Mode.”)

The OR*n* registers are interpreted differently depending on which of the three machine types is selected for that bank.

10.3.1.2.1 Address Mask

The address mask fields of the option registers (OR*n*[AM]) mask up to 17 bits of the corresponding BR*n*[BA] fields. The 15 lsbs of the 32-bit internal 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 10-5 shows memory bank sizes from 256 Kbytes to 4 Gbytes.

Table 10-5. 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*n*)—GPCM Mode

Figure 10-3 shows the bit fields for OR*n* when the corresponding BR*n*[MSEL] selects the GPCM machine.

Offset 0x0_5004 (OR0) 0x0_5024 (OR4) Access: read/write
0x0_500C (OR1) 0x0_502C (OR5)
0x0_5014 (OR2) 0x0_5034 (OR6)
0x0_501C (OR3) 0x0_503C (OR7)

	0–16	17	18	19	20	21	22	23	24–27	28	29	30	31
R/W	AM	—		BCTLD	CSNT	ACS		XACS	SCY	SETA	TRLX	EHTR	EAD
Reset OR0 [1]	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0		0	1	1	1	1	1 1 1 1	0	1	1	1
Reset OR1–OR7	All zeros												

[1] OR0 has this value set during reset (GPCM is the default control machine for all banks coming out of reset). All other option registers have all bits cleared.

Figure 10-3. Option Registers (OR*n*) in GPCM Mode

Table 10-6 describes OR*n* fields for GPCM mode.

Table 10-6. OR*n*—GPCM Field Descriptions

Bits	Name	Description
0–16	AM	GPCM address mask. Masks corresponding BR*n* bits. Masking address bits independently allows external devices of different size address ranges to be used. Address mask bits can be set or cleared in any order in the field, allowing a resource to reside in more than one area of the address map. 0 Corresponding address bits are masked. 1 Corresponding address bits are used in the comparison between base and transaction addresses. See Section 10.3.1.2.1, "Address Mask."
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 $\overline{\text{LCS}n}$ and $\overline{\text{LWE}}$ are negated during an external memory write access handled by the GPCM, provided that ACS ≠ 00 (when ACS = 00, only $\overline{\text{LWE}}$ is affected by the setting of CSNT). This helps meet address/data hold times for slow memories and peripherals. 0 $\overline{\text{LCS}n}$ and $\overline{\text{LWE}}$ are negated normally. 1 $\overline{\text{LCS}n}$ and $\overline{\text{LWE}}$ are negated earlier depending on the value of LCRR[CLKDIV]. **LCRR[CLKDIV] / CSNT / Meaning** x / 0 / $\overline{\text{LCS}n}$ and $\overline{\text{LWE}}$ are negated normally. 2 / 1 / $\overline{\text{LCS}n}$ and $\overline{\text{LWE}}$ are negated normally. 4 or 8 / 1 / $\overline{\text{LCS}n}$ and $\overline{\text{LWE}}$ are negated quarter of a bus clock cycle earlier.
21–22	ACS	Address to chip-select setup. Determines the delay of the $\overline{\text{LCS}n}$ assertion relative to the address change when the external memory access is handled by the GPCM. At system reset, OR0[ACS] = 11 **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. x / 01 / Reserved. 2 / 10 / $\overline{\text{LCS}n}$ is output a half bus clock cycle after the address lines. 2 / 11 / $\overline{\text{LCS}n}$ is output a half bus clock cycle after the address lines. 4 or 8 / 10 / $\overline{\text{LCS}n}$ is output a quarter bus clock cycle after the address lines. 4 or 8 / 11 / $\overline{\text{LCS}n}$ is output a half bus clock cycle after the address lines.
23	XACS	Extra address to chip-select setup. Setting this bit increases the delay of the $\overline{\text{LCS}n}$ assertion relative to the address change when the external memory access is handled by the GPCM. After a system reset, OR0[XACS] = 1. 0 Address to chip-select setup is determined by ORx[ACS] and LCRR[CLKDIV]. 1 Address to chip-select setup is extended (see Table 10-23 and Table 10-24 for LCRR[CLKDIV] = 4 or 8, Table 10-25 and Table 10-26 for LCRR[CLKDIV] = 2).

Table 10-6. OR*n*—GPCM Field Descriptions (continued)

Bits	Name	Description
24–27	SCY	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. 0000 No wait states 0001 1 bus clock cycle wait state ... 1111 15 bus clock cycle wait states
28	SETA	External address termination. 0 Access is terminated internally by the memory controller unless the external device asserts $\overline{\text{LGTA}}$ earlier to terminate the access. 1 Access is terminated externally by asserting the $\overline{\text{LGTA}}$ external signal. (Only $\overline{\text{LGTA}}$ can terminate the access).
29	TRLX	Timing relaxed. Modifies the settings of timing parameters for slow memories or peripherals. 0 Normal timing is generated by the GPCM. 1 Relaxed timing on the following parameters: • Adds an additional cycle between the address and control signals (only if ACS is not equal to 00). • Doubles the number of wait states specified by SCY, providing up to 30 wait states. • Works in conjunction with EHTR to extend hold time on read accesses. • $\overline{\text{LCS}}$*n* (only if ACS is not equal to 00) and $\overline{\text{LWE}}$ signals are negated one cycle earlier during writes.
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.

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	EAD	External address latch delay. Allow extra bus clock cycles when using external address latch (LALE). 0 No additional bus clock cycles (LALE asserted for one bus clock cycle only) 1 Extra bus clock cycles are added (LALE is asserted for the number of bus clock cycles specified by LCRR[EADC]).

10.3.1.2.3 Option Registers (OR*n*)—UPM Mode

Figure 10-4 shows the bit fields for OR*n* when the corresponding BR*n*[MSEL] selects a UPM machine.

Offset 0x0_5004 (OR0), 0x0_500C (OR1), 0x0_5014 (OR2), 0x0_501C (OR3), 0x0_5024 (OR4), 0x0_502C (OR5), 0x0_5034 (OR6), 0x0_503C (OR7) — Access: read/write

	0–16	17–18	19	20–22	23	24–28	29	30	31
R/W	AM	—	BCTLD	—	BI	—	TRLX	EHTR	EAD
Reset	All zeros								

Figure 10-4. Option Registers (OR*n*) in UPM Mode

Table 10-7 describes OR*n* fields for UPM mode.

Table 10-7. OR*n*—UPM Field Descriptions

Bits	Name	Description
0–16	AM	UPM address mask. Masks corresponding BR*n* bits. Masking address bits independently allows external devices of different size address ranges to be used. Address mask bits can be set or cleared in any order in the field, allowing a resource to reside in more than one area of the address map. 0 Corresponding address bits are masked. 1 The corresponding address bits are used in the comparison with address signals.
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	TRLX	Timing relaxed. Works in conjunction with EHTR to extend hold time on read accesses.

Table 10-7. OR*n*—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. 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	EAD	External address latch delay. Allow extra bus clock cycles when using external address latch (LALE). 0 No additional bus clock cycles (LALE asserted for one bus clock cycle only) 1 Extra bus clock cycles are added (LALE is asserted for the number of bus clock cycles specified by LCRR[EADC]).

10.3.1.2.4 Option Registers (OR*n*)—SDRAM Mode

Figure 10-5 shows the bit fields for OR*n* when the corresponding BR*n*[MSEL] selects the SDRAM machine.

Offset 0x0_5004 (OR0), 0x0_500C (OR1), 0x0_5014 (OR2), 0x0_501C (OR3), 0x0_5024 (OR4), 0x0_502C (OR5), 0x0_5034 (OR6), 0x0_503C (OR7) Access: Read/write

	0–16	17–18	19–21	22	23–25	26–27	30	31
R/W	AM	—	COLS	—	ROWS	PMSEL	—	EAD
Reset	All zeros							

Figure 10-5. Option Registers (OR*n*) in SDRAM Mode

Table 10-8 describes OR*n* fields for SDRAM mode.

Table 10-8. OR*n*—SDRAM Field Descriptions

Bits	Name	Description
0–16	AM	SDRAM address mask. Masks corresponding BR*n* bits. Masking address bits independently allows external devices of different size address ranges to be used. Address mask bits can be set or cleared in any order in the field, allowing a resource to reside in more than one area of the address map. AM can be read or written at any time. 0 Corresponding address bits are masked. 1 The corresponding address bits are used in the comparison with address signals.
17–18	—	Reserved
19–21	COLS	Number of column address lines. Sets the number of column address lines in the SDRAM device. 000 7 100 11 001 8 101 12 010 9 110 13 011 10 111 14

Table 10-8. OR*n*—SDRAM Field Descriptions (continued)

Bits	Name	Description
22	—	Reserved
23–25	ROWS	Number of row address lines. Sets the number of row address lines in the SDRAM device. 000 9 100 13 001 10 101 14 010 11 110 15 011 12 111 Reserved
26	PMSEL	Page mode select. Selects page mode for the SDRAM connected to the memory controller bank. 0 Back-to-back page mode (normal operation). Page is closed when the bus becomes idle. 1 Page is kept open until a page miss or refresh occurs.
27–30	—	Reserved
31	EAD	External address latch delay. Allow extra bus clock cycles when using external address latch (LALE). 0 No additional bus clock cycles (LALE asserted for one bus clock cycle only) 1 Extra bus clock cycles are added (LALE is asserted for the number of bus clock cycles specified by LCRR[EADC]).

10.3.1.3 UPM Memory Address Register (MAR)

Figure 10-6 shows the fields of the UPM memory address register (MAR).

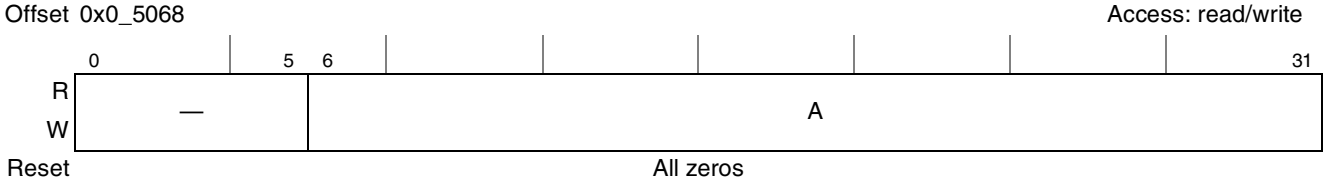


Figure 10-6. UPM Memory Address Register (MAR)

Table 10-9 describes the MAR fields.

Table 10-9. MAR Field Descriptions

Bits	Name	Description
0–5	—	Reserved.
6–31	A	Address that can be output to the address signals under control of the AMX bits in the UPM RAM word.

10.3.1.4 UPM Mode Registers (M*n*MR)

The UPM machine mode registers (MAMR, MBMR, and MCMR), shown in Figure 10-7, contain the configuration for the three UPMs.

Offset 0x0_5070 (MAMR)
0x0_5074 (MBMR)
0x0_5078 (MCMR)

Access: read/write

	0	1	2 3	4	5 7	8 9	10 12	13	14 15	16 17	18 21	22 25	26 31
R W	—	RFEN	OP	UWPL	AM	DS	G0CL	GPL4	RLF	RLF	WLF	TLF	MAD
Reset	All zeros												

Figure 10-7. UPM Mode Registers (M*n*MR)

Table 10-10 describes UPM mode fields.

Table 10-10. M*n*MR Field Descriptions

Bits	Name	Description
0	—	Reserved
1	RFEN	Refresh enable. Indicates that the UPM needs refresh services. This bit must be set for UPMA (refresh executor) if refresh services are required on any UPM assigned chip selects. If MAMR[RFEN] = 0, no refresh services can be provided, even if UPMB and/or UPMC have their RFEN bit set. 0 Refresh services are not required 1 Refresh services are required
2–3	OP	Command opcode. Determines the command executed by the UPM*n* when a memory access hits a UPM assigned bank. 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 signal when in UPM mode. 0 LUPWAIT is active high. 1 LUPWAIT is active low.

Table 10-10. M*n*MR Field Descriptions (continued)

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>5–7</td><td>AM</td><td>Address multiplex size. Determines how the address of the current memory cycle can be output on the address signals. This field is needed when interfacing with devices requiring row and column addresses multiplexed on the same signals.
<table>
<tr><th>Value</th><th>LA0–LA15</th><th>LA16</th><th>LA17</th><th>LA18</th><th>LA19–LA28</th><th>LA29</th><th>LA30</th><th>LA31</th></tr>
<tr><td>000</td><td>0</td><td>A8</td><td>A9</td><td>A10</td><td>A11–A20</td><td>A21</td><td>A22</td><td>A23</td></tr>
<tr><td>001</td><td>0</td><td>A7</td><td>A8</td><td>A9</td><td>A10–A19</td><td>A20</td><td>A21</td><td>A22</td></tr>
<tr><td>010</td><td>0</td><td>A6</td><td>A7</td><td>A8</td><td>A9–A18</td><td>A19</td><td>A20</td><td>A21</td></tr>
<tr><td>011</td><td>0</td><td>A5</td><td>A6</td><td>A7</td><td>A8–A17</td><td>A18</td><td>A19</td><td>A20</td></tr>
<tr><td>100</td><td>0</td><td>A4</td><td>A5</td><td>A6</td><td>A7–A16</td><td>A17</td><td>A18</td><td>A19</td></tr>
<tr><td>101</td><td>0</td><td>A3</td><td>A4</td><td>A5</td><td>A6–A15</td><td>A16</td><td>A17</td><td>A18</td></tr>
<tr><td>110–111</td><td colspan="8">Reserved</td></tr>
</table></td></tr>
<tr><td>8–9</td><td>DS</td><td>Disable timer period. Guarantees a minimum time between accesses to the same memory bank controlled by UPMn. The disable timer is turned on by the TODT bit in the RAM array word, and when expired, the UPMn allows the machine access to handle a memory pattern to the same bank. Accesses to a different bank by the same UPMn 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.
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</td></tr>
<tr><td>10–12</td><td>G0CL</td><td>General line 0 control. Determines which logical address line can be output to the LGPL0 signal when the UPMn is selected to control the memory access.
000 A12
001 A11
010 A10
011 A9
100 A8
101 A7
110 A6
111 A5</td></tr>
<tr><td>13</td><td>GPL4</td><td>LGPL4 output line disable. Determines how the LGPL4/LUPWAIT signal is controlled by the corresponding bits in the UPMn array. See Table 10-30 on page 10-66.
<table>
<tr><th rowspan="2">Value</th><th rowspan="2">LGPL4/LUPWAIT Signal Function</th><th colspan="2">Interpretation of UPM Word Bits</th></tr>
<tr><th>G4T1/DLT3</th><th>G4T3/WAEN</th></tr>
<tr><td>0</td><td>LGPL4 (output)</td><td>G4T1</td><td>G4T3</td></tr>
<tr><td>1</td><td>LUPWAIT (input)</td><td>DLT3</td><td>WAEN</td></tr>
</table></td></tr>
</table>

Table 10-10. M*n*MR Field Descriptions (continued)

Bits	Name	Description
14–17	RLF	Read loop field. Determines the number of times a loop defined in the UPM*n* is executed for a burst- or single-beat read pattern or when M*n*MR[OP] = 11 (run command) 0000 16 0001 1 0010 2 0011 3 ... 1110 14 1111 15
18–21	WLF	Write loop field. Determines the number of times a loop defined in the UPM*n* is executed for a burst- or single-beat write pattern. 0000 16 0001 1 0010 2 0011 3 ... 1110 14 1111 15
22–25	TLF	Refresh loop field. Determines the number of times a loop defined in the UPM*n* is 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*n*.

10.3.1.5 Memory Refresh Timer Prescaler Register (MRTPR)

MRTPR shown in Figure 10-8, is used to divide the system clock to provide the SDRAM and UPM refresh timers clock.

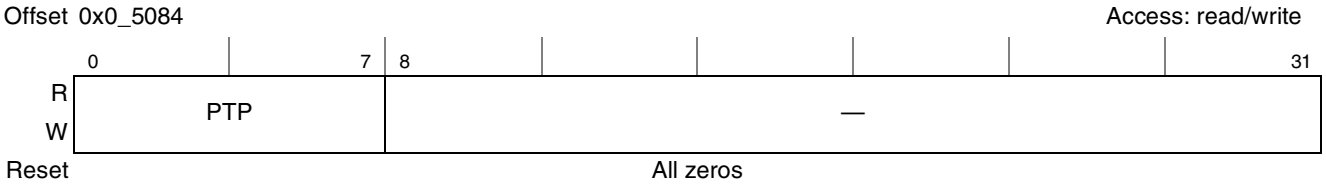


Figure 10-8. Memory Refresh Timer Prescaler Register (MRTPR)

Table 10-11 describes MRTPR fields.

Table 10-11. 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 0x0000_0000, which represents the maximum divider of 256.
8–31	—	Reserved

10.3.1.6 UPM Data Register (MDR)

MDR shown in Figure 10-9, contains data written to or read from the RAM array for UPM read or write commands. MDR must be set up before issuing a write command to the UPM.

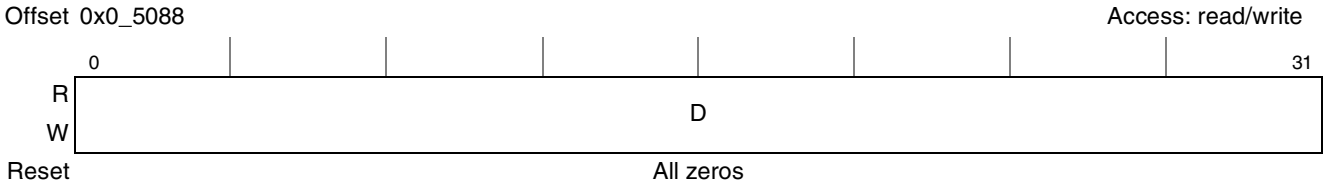


Figure 10-9. UPM Data Register (MDR)

Table 10-12 describes MDR[D].

Table 10-12. MDR Field Description

Bits	Name	Description
0–31	D	The data to be read or written into the RAM array when a write or read command is supplied to the UPM (M*n*MR[OP] = 01 or M*n*MR[OP] = 10).

10.3.1.7 Local Bus SDRAM Machine Mode Register (LSDMR)

LSDMR shown in Figure 10-10, is used to configure operations pertaining to SDRAM.

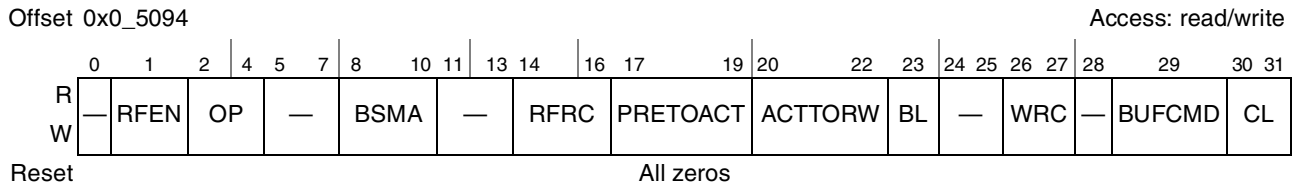


Figure 10-10. Local Bus SDRAM Machine Mode Register (LSDMR)

Table 10-12 describes LSDMR fields.

Table 10-13. LSDMR Field Descriptions

Bits	Name	Description
0	—	Reserved
1	RFEN	Refresh enable. Indicates that the SDRAM requires refresh services. 0 Refresh services are not required. 1 Refresh services are required.

Table 10-13. LSDMR Field Descriptions (continued)

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>2–4</td><td>OP</td><td>SDRAM operation. Selects the operation that occurs when the SDRAM device is accessed. See Section 10.4.3.3, “Intel PC133 and JEDEC-Standard SDRAM Interface Commands.”
<table>
<tr><th>Value</th><th>Meaning</th><th>Use</th></tr>
<tr><td>000</td><td>Normal operation</td><td>Normal operation</td></tr>
<tr><td>001</td><td>Auto refresh</td><td>Initialization</td></tr>
<tr><td>010</td><td>Self refresh</td><td>Debug</td></tr>
<tr><td>011</td><td>Mode register write</td><td>Initialization</td></tr>
<tr><td>100</td><td>Precharge bank</td><td>Debug</td></tr>
<tr><td>101</td><td>Precharge all banks</td><td>Initialization</td></tr>
<tr><td>110</td><td>Activate bank</td><td>Debug</td></tr>
<tr><td>111</td><td>Read/write without valid data transfer</td><td>Debug</td></tr>
</table></td></tr>
<tr><td>5–7</td><td>—</td><td>Reserved</td></tr>
<tr><td>8–10</td><td>BSMA</td><td>Bank select multiplexed address line. Selects which address signals serve as the 2-bit bank-select address for SDRAM. Note that only 4-bank SDRAMs are supported.
000 LA[12:13] 100 LA[16:17]
001 LA[13:14] 101 LA[17:18]
010 LA[14:15] 110 LA[18:19]
011 LA[15:16] 111 LA[19:20]</td></tr>
<tr><td>11–13</td><td>—</td><td>Reserved</td></tr>
<tr><td>14–16</td><td>RFRC</td><td>Refresh recovery. Sets the refresh recovery interval in bus clock cycles. Defines the earliest timing for an ACTIVATE or REFRESH command after a REFRESH command. See Section 10.4.3.7.5, “Refresh Recovery Interval (RFRC).”
000 Reserved 100 6 clocks
001 3 clocks 101 7 clocks
010 4 clocks 110 8 clocks
011 5 clocks 111 16 clocks</td></tr>
<tr><td>17–19</td><td>PRETOACT</td><td>Defines the earliest timing for ACTIVATE or REFRESH command after a PRECHARGE command (number of bus clock cycle wait states). See Section 10.4.3.7.1, “Precharge-to-Activate Interval.”
000 8 100 4
001 1 101 5
010 2 110 6
011 3 111 7</td></tr>
<tr><td>20–22</td><td>ACTTORW</td><td>Defines the earliest timing for READ/WRITE command after an ACTIVATE command (number of bus clock cycle wait states). See Section 10.4.3.7.2, “Activate-to-Read/Write Interval.”
000 8 100 4
001 Reserved 101 5
010 2 110 6
011 3 111 7</td></tr>
<tr><td>23</td><td>BL</td><td>Sets the burst length for SDRAM accesses.
0 SDRAM burst length is 4. Use this value if the device port size is 16.
1 SDRAM burst length is 8. Use this value if the device port size is 32 or 8.</td></tr>
</table>

Table 10-13. LSDMR Field Descriptions (continued)

Bits	Name	Description
24–25	—	Reserved
26–27	WRC	Write recovery time. Defines the earliest timing for PRECHARGE command after the last data is written to the SDRAM. See Section 10.4.3.7.3, “Column Address to First Data Out—CAS Latency.” 00 4 01 Reserved 10 2 11 3
28	—	Reserved
29	BUFCMD	Control line assertion timing. If external buffers are placed on the control lines going to both the SDRAM and address lines, setting BUFCMD causes all SDRAM control lines except $\overline{\text{LCS}n}$, LCKE, LALE and $\overline{\text{LSDDQM}}$[0:3] to be asserted for LCRR[BUFCMDC] cycles, instead of one. See Section 10.4.3.7.6, “External Address and Command Buffers (BUFCMD).” 0 Normal timing for the control lines 1 All control lines except $\overline{\text{LCS}n}$ are asserted for the number of cycles specified by LCRR[BUFCMDC].
30–31	CL	CAS latency. Defines the timing for first read data after SDRAM samples a column address. 00 Extended CAS latency. According to LCRR[ECL]. See Table 10-22. 01 1 10 2 11 3

10.3.1.8 UPM Refresh Timer (LURT)

LURT, shown in Figure 10-11, generates a refresh request for all valid banks that selected a UPM machine and are refresh-enabled (M*n*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.

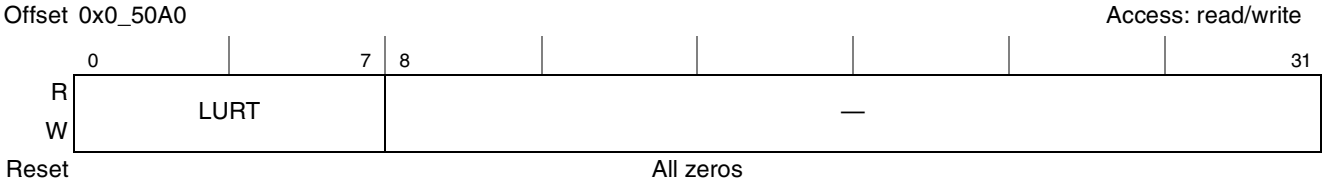


Figure 10-11. UPM Refresh Timer (LURT)

Table 10-14 describes LURT fields.

Table 10-14. LURT Field Descriptions

Bits	Name	Description
0–7	LURT	UPM refresh timer period. Determines, along with the timer prescaler (MRTPR), the timer period according to the following equation: $\text{TimerPeriod} = \frac{\text{LURT}}{\left(\frac{\text{Fsystemclock}}{\text{MRTPR[PTP]}}\right)}$ 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.
8–31	—	Reserved

10.3.1.9 SDRAM Refresh Timer (LSRT)

LSRT shown in Figure 10-12, generates a refresh request for all valid banks that selected a SDRAM machine and are refresh-enabled (LSDMR[RFEN] = 1). Each time the timer expires, all qualifying banks generate a bank staggering auto-refresh request using the SDRAM machine.

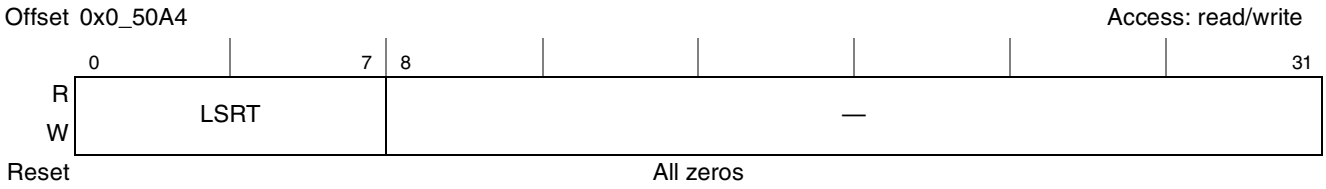


Figure 10-12. LSRT SDRAM Refresh Timer (LSRT)

Table 10-15 describes LSRT fields.

Table 10-15. LSRT Field Descriptions

Bits	Name	Description
0–7	LSRT	SDRAM refresh timer period. Determines, along with the timer prescaler (MRTPR), the timer period according to the following equation: $\text{TimerPeriod} = \frac{\text{LSRT}}{\left(\frac{\text{Fsystemclock}}{\text{MRTPR[PTP]}}\right)}$ **Example:** For a 266-MHz system clock and a required service rate of 15.6 µs, given PTP = 32, the LSRT 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.
8–31	—	Reserved

10.3.1.10 Transfer Error Status Register (LTESR)

The LBC has five registers for error management.

- The transfer error status register (LTESR) indicates the cause of an error.
- The transfer error check disable register (LTEDR) is used to enable (and disable) error checking.
- The transfer error check interrupt register (LTEIR) enables reporting of errors through an interrupt.
- The transfer error attributes register (LTEATR) captures source attributes of an error.
- The transfer error address register (LTEAR) captures the address of a transaction that caused an error.

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, 0b0400_0000 should be written to the register.

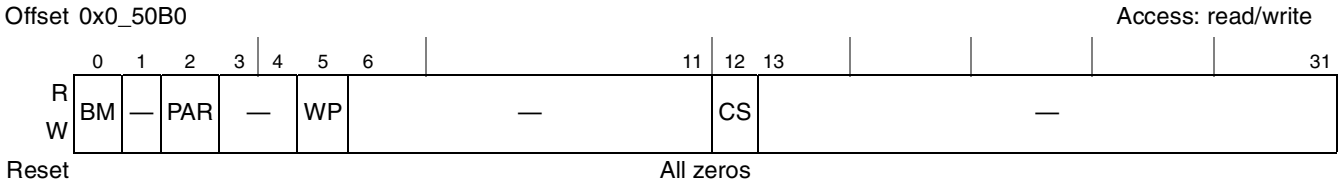


Figure 10-13. Transfer Error Status Register (LTESR)

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 8 bus clock cycles from the start of a transaction.
1	—	Reserved
2	PAR	Parity 0 No local bus parity error. 1 Local bus parity error. LTEATR[PB] indicates the byte lane that caused the error and LTEATR[BNK] indicates which memory controller bank was accessed.
3–4	—	Reserved
5	WP	Write protect error 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 LBC that did not hit any memory bank.
13–31	—	Reserved

10.3.1.11 Transfer Error Check Disable Register (LTEDR)

LTEDR shown in Figure 10-14, is used to disable error checking. Note that control of error checking is independent of control of reporting of errors (LTEIR) through the interrupt mechanism.

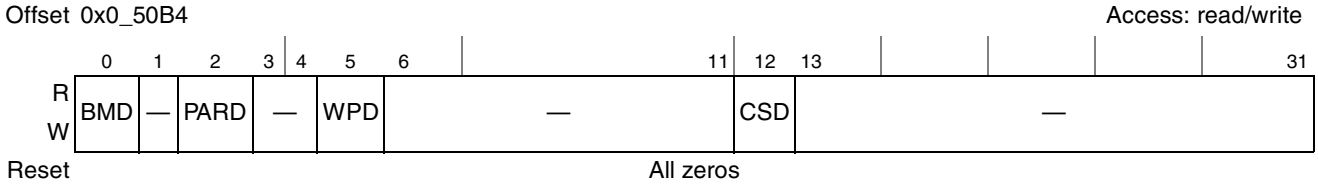


Figure 10-14. Transfer Error Check Disable Register (LTEDR)

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.
1	—	Reserved
2	PARD	Parity error checking disabled. 0 Parity error checking is enabled. 1 Parity 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.
13–31	—	Reserved

10.3.1.12 Transfer Error Interrupt Enable Register (LTEIR)

LTEIR shown in Figure 10-15, is used to send or block error reporting through the LBC internal interrupt mechanism. Software should clear pending errors in LTESR before enabling interrupts. After an interrupt has occurred, clearing relevant LTESR error bits negates the interrupt.

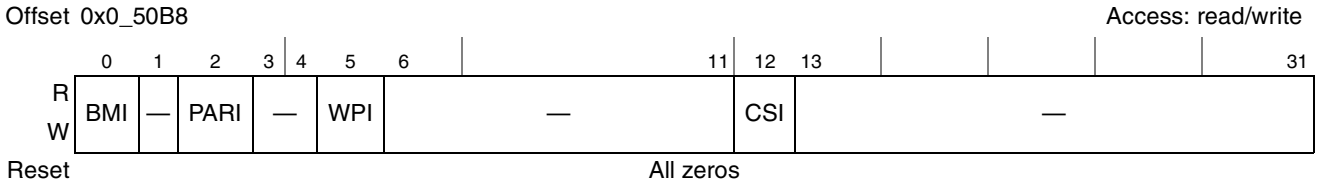


Figure 10-15. Transfer Error Interrupt Enable Register (LTEIR)

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	—	Reserved
2	PARI	Parity error interrupt enable. 0 Parity error reporting is disabled. 1 Parity 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.
13–31	—	Reserved

10.3.1.13 Transfer Error Attributes Register (LTEATR)

LTEATR is shown in Figure 10-16. After LTEATR[V] has been set, software must clear this bit to allow LTEATR and LTEAR to update following any subsequent errors.

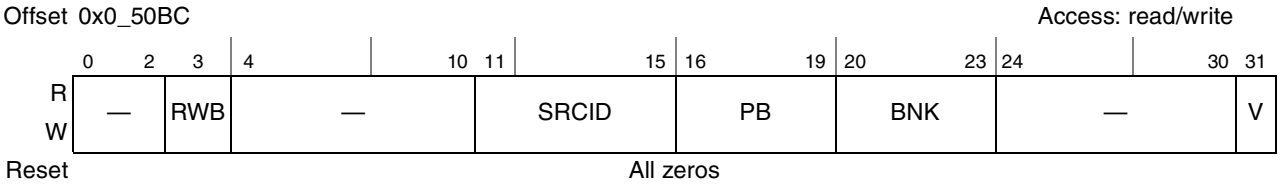


Figure 10-16. Transfer Error Attributes Register (LTEATR)

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 LBC. The coding of the source ID debug information is the same as the coding of AEATR[MSTR_ID] (see Section 6.2.6, “Arbiter Event Attributes Register (AEATR).”)

Table 10-19. LTEATR Field Descriptions (continued)

Bits	Name	Description
16–19	PB	Parity error on byte. There are four parity error status bits, one per byte lane. A bit is set for the byte that had a parity error (bit 16 represents byte 0, the most significant byte lane).
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. Note that BNK is invalid if the error was not caused by parity checks.
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.

10.3.1.14 Transfer Error Address Register (LTEAR)

The transfer error address register (LTEAR) is shown in Figure 10-17.

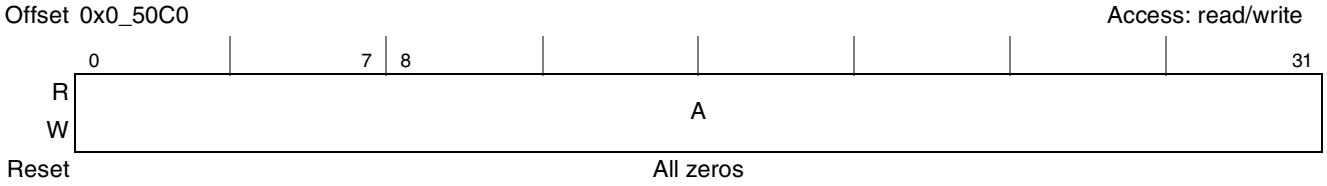


Figure 10-17. Transfer Error Address Register (LTEAR)

Table 10-20 describes LTEAR[A].

Table 10-20. LTEAR Field Descriptions

Bits	Name	Description
0–31	A	Transaction address for the error. Holds the 32-bit address of the transaction resulting in an error.

10.3.1.15 Local Bus Configuration Register (LBCR)

LBCR is shown in Figure 10-18.

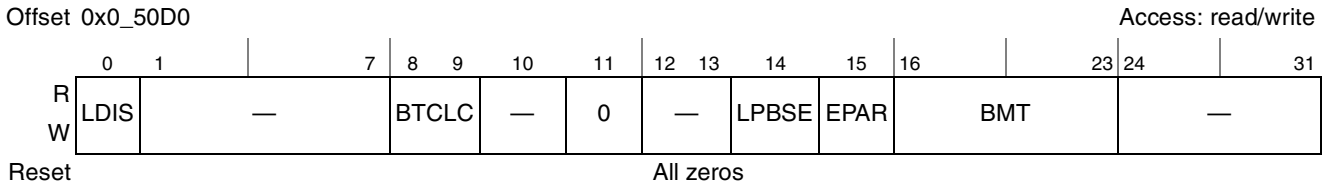


Figure 10-18. Local Bus Configuration Register

Table 10-21 describes LBCR fields.

Table 10-21. 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.
10–13	—	Reserved
14	LPBSE	Enables parity byte select on $\overline{LGTA}$/LGPL4/LUPWAIT/LPBSE signal. 0 Parity byte select is disabled. $\overline{LGTA}$/LGPL4/LUPWAIT/LPBSE signal is available for memory control as LGPL4 (output) or $\overline{LGTA}$/LUPWAIT (input). 1 Parity byte select is enabled. $\overline{LGTA}$/LGPL4/LUPWAIT/LPBSE signal is dedicated as the parity byte select output, and $\overline{LGTA}$/LUPWAIT is disabled.
15	EPAR	Determines odd or even parity. Writing the memory with EPAR = 1 and reading the memory with EPAR = 0 generates parity errors for testing. 0 Odd parity 1 Even parity
16–23	BMT	Bus monitor timing. Defines the bus monitor time-out period. Clearing BMT (reset value) selects the maximum count of 2048 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 x 8. Apart from BMT = 0x00, the minimum value of BMT is 5, corresponding with 40 bus cycles. Shorter time-outs may result in spurious errors during SDRAM operation. See Section 10.4.1.5, “Parity Generation and Checking (LDP),”
24–31	—	Reserved

10.3.1.16 Clock Ratio Register (LCRR)

The clock ratio register, shown inFigure 10-19, sets the system clock to LBC bus frequency ratio. It also provides configuration bits for extra delay cycles for address and control signals.

NOTE

For proper operation of the system, this register setting must not be altered while local bus memories or devices are being accessed. Special care needs to be taken when running instructions from an local bus controller memory.

Offset 0x0_50D4 Access: read/write

	0	1	2	4	5	6	7	8	13	14	15	16	27	28	31
R W	DBYP	—	BUFCMDC	—		ECL		—		EADC		—		CLKDIV	
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Figure 10-19. Clock Ratio Register (LCRR)

Table 10-22 describes LCRR fields.

Table 10-22. LCRR Field Descriptions

Bits	Name	Description
0	DBYP	DLL bypass. Should be set when using low bus clock frequencies if the DLL is unable to lock. When in DLL bypass mode, incoming data is captured in the middle of the bus clock cycle. 0 The DLL is enabled. 1 The DLL is bypassed (default).
1	—	Reserved
2–3	BUFCMDC	Additional delay cycles for SDRAM control signals. Defines the number of cycles to be added for each SDRAM command when LSDMR[BUFCMD] = 1. 00 4 01 1 10 2 11 3
4–5	—	Reserved
6–7	ECL	Extended CAS latency. Determines the extended CAS latency for SDRAM accesses when LSDMR[CL] = 00. 00 4 01 5 10 6 11 7
8–13	—	Reserved
14–15	EADC	External address delay cycles.Defines the number of cycles for the assertion of LALE. 00 4 01 1 10 2 11 3
16–27	—	Reserved
28–31	CLKDIV	System (input) clock divider. Sets the frequency ratio between the (input) system clock and the memory bus clock. Only the values shown in the table below are allowed. 0000–0001 reserved 0010 2 0011 reserved 0100 4 0101–0111 reserved 1000 8 1001–1111 reserved

10.4 Functional Description

The LBC allows the implementation of memory systems with very specific timing requirements.

- The SDRAM machine provides an interface to SDRAMs using bank interleaving and back-to-back page mode to achieve high performance through a multiplexed address/data bus. An internal DLL for bus clock generation ensures improved data set-up margins for board designs.
- 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 and access to low-performance memory-mapped peripherals.

- 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. 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.

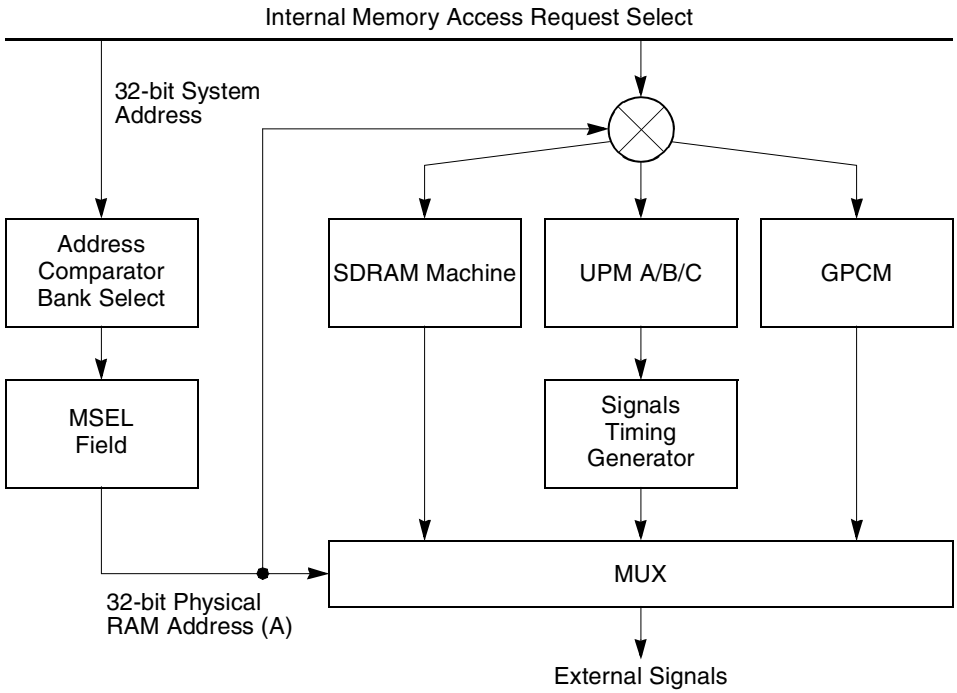


Figure 10-20. Basic Operation of Memory Controllers in the LBC

Each memory bank (chip select) can be assigned to any one of these three type of machines through the machine select bits of the base register for that bank (BR*n*[MSEL]), as illustrated in Figure 10-20. If a bank match occurs, the corresponding machine (GPCM, SDRAM 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

These subsections describe the basic architecture of the LBC.

10.4.1.1 Address and Address Space Checking

The defined base addresses are written to the BR*n* registers; 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

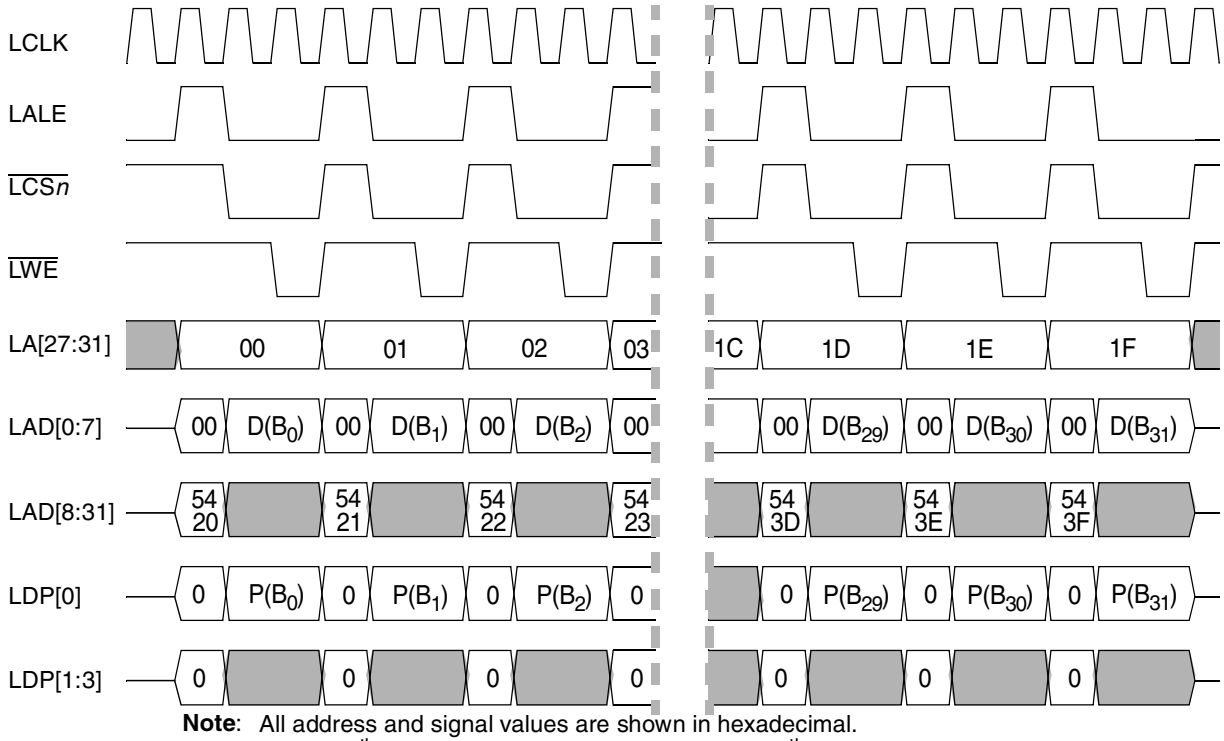
attributes defined in the BR*n* and OR*n* for that bank are used to control the memory access. If a match is found in more than one bank, the lowest-numbered bank handles the memory access (that is, bank 0 has priority over bank 1).

10.4.1.2 External Address Latch Enable Signal (LALE)

The local bus uses a multiplexed address/data bus. Therefore the LBC must distinguish between address and data phases, which take place on the same bus (LAD[0:31] signals). The LALE signal, when asserted, signifies an address phase during which the LBC drives the memory address on the LAD[0:31] signals. An external address latch uses this signal to capture the address and provide it to the address signals of the memory or peripheral device. When LALE is negated, LAD[0:31] then serves as the (bi-directional) data bus for the access. Any address phase initiates the assertion of LALE, which has a programmable duration of between 1 and 4 bus clock cycles.

The frequency of LALE assertion varies across the three memory controllers. For GPCM, every assertion of $\overline{\text{LCS}}n$ is considered an independent access, and accordingly, LALE asserts prior to each such access. For example, GPCM driving an 8-bit port would assert LALE and $\overline{\text{LCS}}n$ 32 times in order to satisfy a 32-byte cache line transfer.The SDRAM controller asserts LALE only to initiate a burst transfer with a starting address, therefore no more than one assertion of LALE may be required for SDRAM to transfer a 32-byte cache line via a 32-bit port. In the case of UPM, the frequency of LALE assertion depends on how the UPM RAM is programmed. UPM single accesses typically assert LALE once, upon commencement, but it is possible to program UPM to assert LALE several times, and to change the values of LA[27:31] with and without LALE being involved. In general, when using the GPCM and SDRAM controllers it is not necessary to use LA[27:31] if a sufficiently wide latch is used to capture the entire address during LALE phases. UPM may require LA[27:31] if the LBC is generating its own burst address sequence.

To illustrate how a large transaction is handled by the LBC, Figure 10-21 shows LBC signals for GPCM performing a 32-byte write starting at address 0x5420. Note that during each of the 32 assertions of LALE, LA[27:31] exactly mirror LAD[27:31], but during data phases, only LAD[0:7] and LDP0 are driven with valid data and parity, respectively.



Note: All address and signal values are shown in hexadecimal.
D(Bk) = kth of 32 data bytes, P(Bk) = parity bit of kth data byte.

Figure 10-21. Example of 8-Bit GPCM Writing 32 Bytes to Address 0x5420

10.4.1.3 Data Transfer Acknowledge (TA)

The three memory controllers in the LBC generate an internal transfer acknowledge signal, TA, to allow data on LAD[0:31] 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 LBC asserts TA internally. In LBC debug mode, TA is also visible externally on the LDVAL signal. GPCM and SDRAM controllers automatically generate TA according to the timing parameters programmed for them in option and mode registers; a UPM generates TA only when a UPM pattern has the UTA RAM word bit set. Figure 10-22 shows LALE, TA (internal), and $\overline{\text{LCS}n}$. Note that TA and LALE are never asserted together, and that for the duration of LALE, $\overline{\text{LCS}n}$ (or any other control signal) remains negated or frozen.

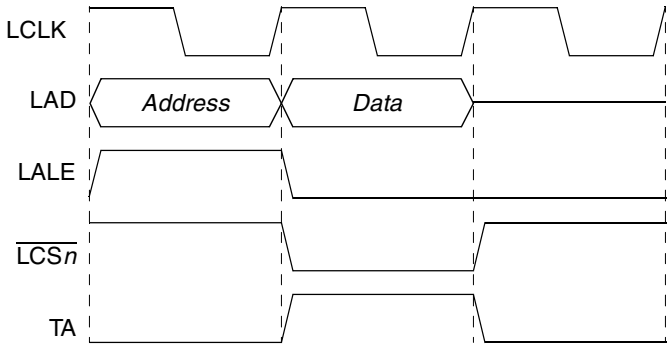


Figure 10-22. Basic LBC Bus Cycle with LALE, TA, and $\overline{\text{LCS}n}$

10.4.1.4 Data Buffer Control (LBCTL)

The memory controller provides a data buffer control signal for the local bus (LBCTL). This signal is activated when a GPCM or UPM controlled bank is accessed. LBCTL can be disabled by setting OR*n*[BCTLD]. Access to an SDRAM machine controlled bank does not activate the LBCTL control. LBCTL can be further configured by LBCR[BCTLC] to act as an extra $\overline{\text{LWE}}$ or an extra $\overline{\text{LOE}}$ signal when in GPCM mode.

If LBCTL is configured as a data buffer control (LBCR[BCTLC] = 00), the signal is asserted (high) on the rising edge of the bus clock on the first cycle of the memory controller operation, coincident with LALE. If the access is a write, LBCTL remains high for the whole duration. However, if the access is a read, LBCTL is negated (low) with the negation of LALE so that the memory device is able to drive the bus. If back-to-back read accesses are pending, LBCTL is asserted (high) one bus clock cycle before the next transaction starts (that is, one bus clock cycle before LALE) to allow a whole bus cycle for the bus to turn around before the next address is driven.

If an external bus transceiver is used, LBCTL should be used to signify the write direction when high. Note that the default (reset and bus idle) value of LBCTL is also high.

10.4.1.5 Parity Generation and Checking (LDP)

Parity can be configured for any bank by programming BR*n*[DECC]. Parity is generated and checked on a per-byte basis using LDP[0:3] for the bank if BR*n*[DECC] = 01 (normal parity) or BR*n*[DECC] = 10 for read-modify-write (RMW) parity. Byte lane parity on LDP[0:3] is generated regardless of the BR*n*[DECC] setting. Note that RMW parity can be used only for 32-bit port size banks. LBCR[EPAR] determines the global type of parity (odd or even).

10.4.1.6 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]) until a data beat is acknowledged on the bus. It then reloads the time-out value and resumes the countdown until the data tenure completes and then idles if there is no pending transaction.Setting LTEDR[BMD] disables bus monitor error checking (i.e. the LTESR[BM] bit is not set by a bus monitor time-out); however, the bus monitor is still active and can generate a UPM exception (as noted in Section 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—particularly for SDRAMs— resulting in incomplete data transfers. Accordingly, apart from the reset value of 0x00 (corresponding with the maximum time-out of 2048 bus cycles), LBCR[BMT] must not be set below 0x05 (or 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, FEPROM, ROM devices, and external peripherals. The GPCM contains two basic configuration register groups—BR*n* and OR*n*.

Figure 10-23 shows a simple connection between an 8-bit port size SRAM device and the LBC in GPCM mode. Byte-write enable signals ($\overline{\text{LWE}}$) are available for each byte written to memory. Also, the output enable signal ($\overline{\text{LOE}}$) is provided to minimize external glue logic. On system reset, a global (boot) chip-select is available that provides a boot ROM chip-select ($\overline{\text{LCS0}}$) before to the system is fully configured.

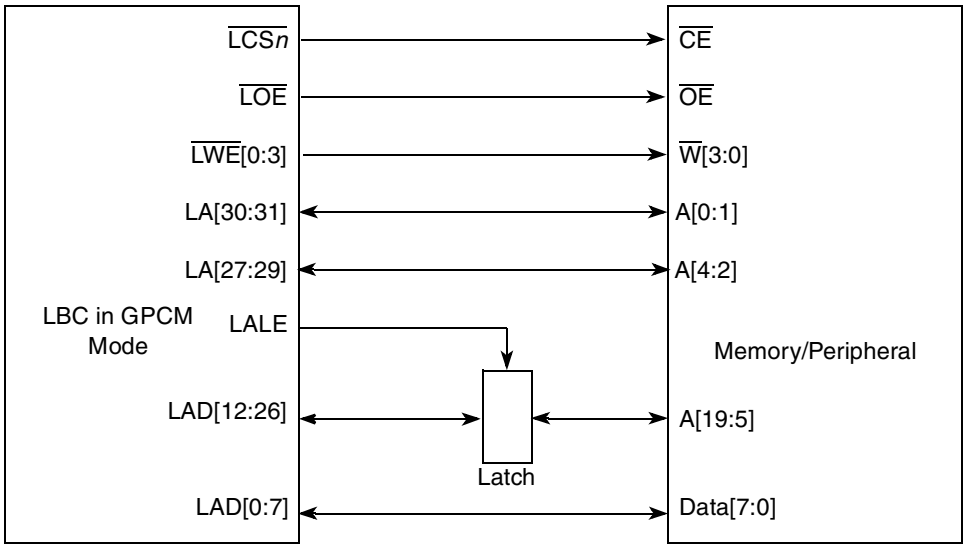


Figure 10-23. Local Bus to GPCM Device Interface

Figure 10-24 shows $\overline{\text{LCS}}$ as defined by the setup time required between the address lines and $\overline{\text{CE}}$. The user can configure OR*n*[ACS] to specify $\overline{\text{LCS}}$ to meet this requirement.

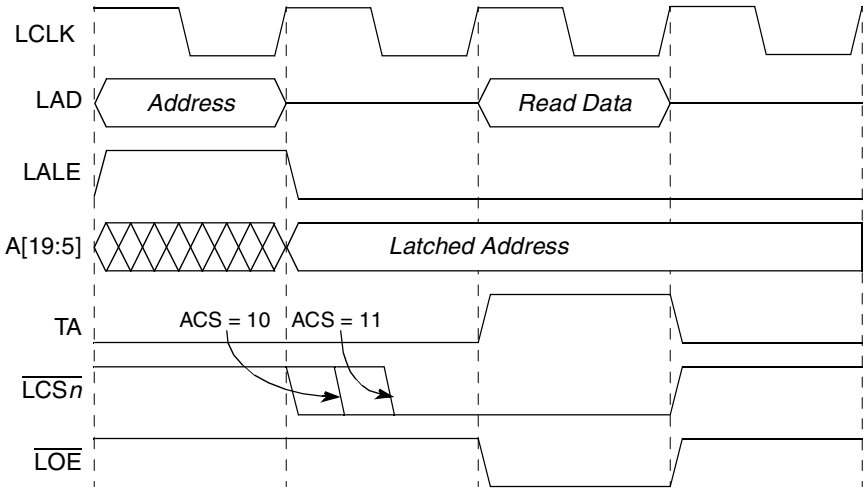


Figure 10-24. GPCM Basic Read Timing (XACS = 0, ACS = 1x, TRLX = 0, CLKDIV = 4,8)

10.4.2.1 Timing Configuration

If BR*n*[MSEL] selects the GPCM, the attributes for the memory cycle are taken from OR*n*. These attributes include the CSNT, ACS, XACS, SCY, TRLX, EHTR and SETA fields. Table 10-23 shows signal behavior and system response for a write access with LCRR[CLKDIV] = 4 or LCRR[CLKDIV] = 8. Table 10-24 shows the signal behavior and system response for a read access with LCRR[CLKDIV] = 4 or LCRR[CLKDIV] = 8. Table 10-25 and Table 10-26 show the write and read signal behavior, respectively, when LCRR[CLKDIV] = 2.

Table 10-23. GPCM Write Control Signal Timing for LCRR[CLKDIV] = 4 or 8

Option Register Attributes				Signal Behavior (Bus Clock Cycles)			
TRLX	XACS	ACS	CSNT	Address to $\overline{\text{LCS}n}$ Asserted	$\overline{\text{LCS}n}$ Negated to Address Change	$\overline{\text{LWE}}$ Negated to Address/Data Invalid	Total Cycles[1]
0	0	00	0	0	0	0	3+SCY
0	0	10	0	1/4	0	0	3+SCY
0	0	11	0	1/2	0	0	3+SCY
0	1	00	0	0	0	0	3+SCY
0	1	10	0	1	0	0	3+SCY
0	1	11	0	2	0	0	4+SCY
0	0	00	1	0	0	–1/4	3+SCY
0	0	10	1	1/4	–1/4	–1/4	3+SCY
0	0	11	1	1/2	–1/4	–1/4	3+SCY
0	1	00	1	0	0	–1/4	3+SCY
0	1	10	1	1	–1/4	–1/4	3+SCY
0	1	11	1	2	–1/4	–1/4	4+SCY
1	0	00	0	0	0	0	3+2*SCY
1	0	10	0	1+1/4	0	0	4+2*SCY
1	0	11	0	1+1/2	0	0	4+2*SCY
1	1	00	0	0	0	0	3+2*SCY
1	1	10	0	2	0	0	4+2*SCY
1	1	11	0	3	0	0	5+2*SCY
1	0	00	1	0	0	–1–1/4	4+2*SCY
1	0	10	1	1+1/4	–1–1/4	–1–1/4	5+2*SCY
1	0	11	1	1+1/2	–1–1/4	–1–1/4	5+2*SCY
1	1	00	1	0	0	–1–1/4	4+2*SCY
1	1	10	1	2	–1–1/4	–1–1/4	5+2*SCY
1	1	11	1	3	–1–1/4	–1–1/4	6+2*SCY

[1] Total cycles when LALE is asserted for one cycle only (OR*n*[EAD] = 0; OR*n*[EAD] = 1 and LCRR[EADC] = 01). Asserting LALE for more than one cycle increases the total cycle count accordingly.

Table 10-24. GPCM Read Control Signal Timing for LCRR[CLKDIV] = 4 or 8

Option Register Attributes				Signal Behavior (bus clock cycles)		
TRLX	EHTR	XACS	ACS	Address to $\overline{\text{LCS}n}$ Asserted	$\overline{\text{LCS}n}$ Negated to Address Change	Total Cycles[1]
0	0	0	00	0	1	4+SCY
0	0	0	10	1/4	1	4+SCY
0	0	0	11	1/2	1	4+SCY
0	0	1	00	0	1	4+SCY
0	0	1	10	1	1	4+SCY
0	0	1	11	2	1	5+SCY
0	1	0	00	0	2	5+SCY
0	1	0	10	1/4	2	5+SCY
0	1	0	11	1/2	2	5+SCY
0	1	1	00	0	2	5+SCY
0	1	1	10	1	2	5+SCY
0	1	1	11	2	2	6+SCY
1	0	0	00	0	5	8+2*SCY
1	0	0	10	1+1/4	5	9+2*SCY
1	0	0	11	1+1/2	5	9+2*SCY
1	0	1	00	0	5	8+2*SCY
1	0	1	10	2	5	9+2*SCY
1	0	1	11	3	5	10+2*SCY
1	1	0	00	0	9	12+2*SCY
1	1	0	10	1+1/4	9	13+2*SCY
1	1	0	11	1+1/2	9	13+2*SCY
1	1	1	00	0	9	12+2*SCY
1	1	1	10	2	9	13+2*SCY
1	1	1	11	3	9	14+2*SCY

[1] Total cycles when LALE is asserted for one cycle only (OR*n*[EAD] = 0; OR*n*[EAD] = 1 and LCRR[EADC] = 01). Asserting LALE for more than one cycle increases the total cycle count accordingly.

Table 10-25. GPCM Write Control Signal Timing for LCRR[CLKDIV] = 2

Option Register Attributes				Signal Behavior (Bus Clock Cycles)			
TRLX	XACS	ACS	CSNT	Address to $\overline{\text{LCSn}}$ Asserted	$\overline{\text{LCS}n}$ Negated to Address Change	$\overline{\text{LWE}}$ Negated to Address/Data Invalid	Total Cycles[1]
0	0	00	0	0	0	0	3+SCY
0	0	10	0	1/2	0	0	3+SCY
0	0	11	0	1/2	0	0	3+SCY
0	1	00	0	0	0	0	3+SCY
0	1	10	0	1	0	0	3+SCY
0	1	11	0	2	0	0	4+SCY
0	0	00	1	0	0	0	3+SCY
0	0	10	1	1/2	0	0	3+SCY
0	0	11	1	1/2	0	0	3+SCY
0	1	00	1	0	0	0	3+SCY
0	1	10	1	1	0	0	3+SCY
0	1	11	1	2	0	0	4+SCY
1	0	00	0	0	0	0	3+2*SCY
1	0	10	0	1+1/2	0	0	4+2*SCY
1	0	11	0	1+1/2	0	0	4+2*SCY
1	1	00	0	0	0	0	3+2*SCY
1	1	10	0	2	0	0	4+2*SCY
1	1	11	0	3	0	0	5+2*SCY
1	0	00	1	0	0	–1	4+2*SCY
1	0	10	1	1+1/2	–1	–1	5+2*SCY
1	0	11	1	1+1/2	–1	–1	5+2*SCY
1	1	00	1	0	0	–1	4+2*SCY
1	1	10	1	2	–1	–1	5+2*SCY
1	1	11	1	3	–1	–1	6+2*SCY

[1] Total cycles when LALE is asserted for one cycle only (OR*n*[EAD]=0; OR*n*[EAD]=1 and LCRR[EADC]=01). Asserting LALE for more than one cycle increases the total cycle count accordingly.

Table 10-26. GPCM Read Control Signal Timing for LCRR[CLKDIV] = 2

Option Register Attributes				Signal Behavior (Bus Clock cycles)		
TRLX	EHTR	XACS	ACS	Address to $\overline{\text{LCS}n}$ Asserted	$\overline{\text{LCS}n}$ Negated to Address Change	Total Cycles[1]
0	0	0	00	0	1	4+SCY
0	0	0	10	1/2	1	4+SCY
0	0	0	11	1/2	1	4+SCY
0	0	1	00	0	1	4+SCY
0	0	1	10	1	1	4+SCY
0	0	1	11	2	1	5+SCY
0	1	0	00	0	2	5+SCY
0	1	0	10	1/2	2	5+SCY
0	1	0	11	1/2	2	5+SCY
0	1	1	00	0	2	5+SCY
0	1	1	10	1	2	5+SCY
0	1	1	11	2	2	6+SCY
1	0	0	00	0	5	8+2*SCY
1	0	0	10	1+1/2	5	9+2*SCY
1	0	0	11	1+1/2	5	9+2*SCY
1	0	1	00	0	5	8+2*SCY
1	0	1	10	2	5	9+2*SCY
1	0	1	11	3	5	10+2*SCY
1	1	0	00	0	9	12+2*SCY
1	1	0	10	1+1/2	9	13+2*SCY
1	1	0	11	1+1/2	9	13+2*SCY
1	1	1	00	0	9	12+2*SCY
1	1	1	10	2	9	13+2*SCY
1	1	1	11	3	9	14+2*SCY

[1] Total cycles when LALE is asserted for one cycle only (OR*n*[EAD]=0; OR*n*[EAD]=1 and LCRR[EADC]=01). Asserting LALE for more than one cycle increases the total cycle count accordingly.

10.4.2.2 Chip-Select Assertion Timing

The banks selected to work with the GPCM support an option to drive the $\overline{\text{LCS}n}$ signal with different timings (with respect to the external address/data bus). $\overline{\text{LCS}n}$ can be driven in any of the following ways:

- Simultaneous with the latched memory address. (This refers to the externally latched address and not the address timing on LAD[0:31]. That is, the chip select does not assert during LALE).

- 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*n*[XACS] = 1.
- Two clock cycles later (for LCRR[CLKDIV] = 2, 4, or 8), when OR*n*[XACS] = 1.
- Three clock cycles later (for LCRR[CLKDIV] = 2, 4, or 8), when OR*n*[XACS] = 1 and OR*n*[TRLX] = 1.

Figure 10-24 shows two chip-select assertion timings for the case LCRR[CLKDIV] = 4 or 8. If LCRR[CLKDIV] = 2, $\overline{\text{LCS}n}$ asserts identically for OR*n*[ACS] = 10 or 11.

10.4.2.2.1 Programmable Wait State Configuration

The GPCM supports internal generation of transfer acknowledge. It allows between 0 and 30 wait states to be added to an access by programming OR*n*[SCY] and OR*n*[TRLX]. Internal generation of transfer acknowledge is enabled if OR*n*[SETA] = 0. If $\overline{\text{LGTA}}$ 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 $\overline{\text{LGTA}}$; otherwise it is terminated by the expiration of the wait state counter. Regardless of the setting of OR*n*[SETA], wait states prolong the assertion duration of both $\overline{\text{LOE}}$ and $\overline{\text{LWE}n}$ in the same manner. When TRLX = 1, the number of wait states inserted by the memory controller is doubled from OR*n*[SCY] cycles to 2*OR*n*[SCY] cycles, allowing a maximum of 30 wait states.

10.4.2.2.2 Chip-Select and Write Enable Negation Timing

Figure 10-23 shows a basic connection between the local bus and a static memory device. In this case, $\overline{\text{LCS}n}$ is connected directly to $\overline{\text{CE}}$ of the memory device. $\overline{\text{LWE}}$[0:3] are connected to the respective $\overline{\text{WE}}$[3:0] signals on the memory device where each $\overline{\text{LWE}}$[0:3] signal corresponds to a different data byte.

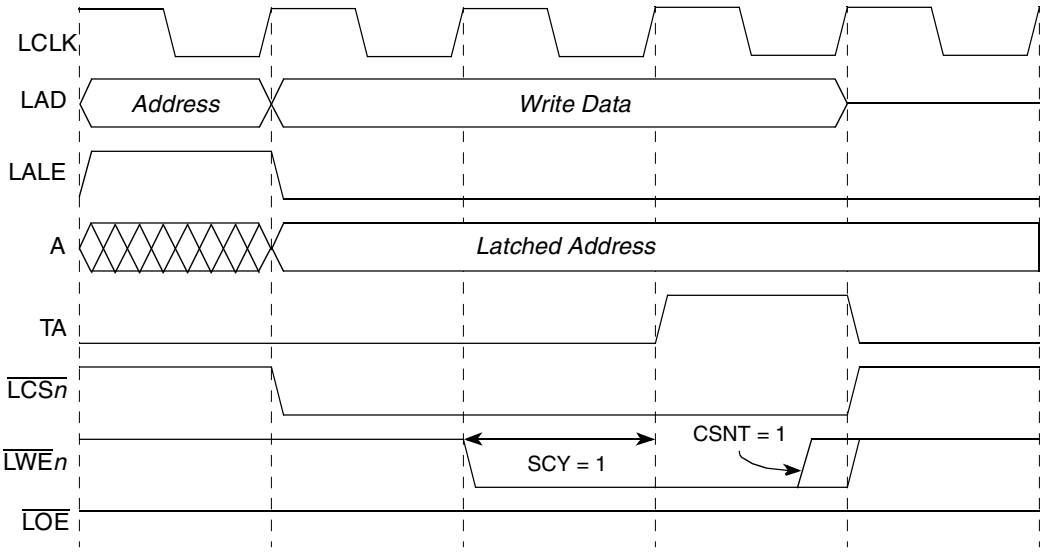


Figure 10-25. GPCM Basic Write Timing (XACS = 0, ACS = 00, CSNT = 1, SCY = 1, TRLX = 0, CLKDIV = 4, 8)

As Figure 10-25 shows, the timing for $\overline{\text{LCS}n}$ is the same as for the latched address. The strobes for the transaction are supplied by $\overline{\text{LOE}}$ or $\overline{\text{LWE}n}$, depending on the transaction direction—read or write (write case shown in the figure). OR*n*[CSNT] controls 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[CLDIV] = 4 or 8. For example, when ACS = 00 and CSNT = 1, $\overline{\text{LWE}n}$ is negated one quarter of a clock earlier, as shown in Figure 10-25. If LCRR[CLDIV] = 2, $\overline{\text{LWE}n}$ is negated either coincident with $\overline{\text{LCS}n}$ or one cycle earlier.

10.4.2.2.3 Relaxed Timing

OR*n*[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.
- $\overline{\text{LCS}n}$ signals are negated 1-cycle earlier during writes (if ACS ≠00).
- $\overline{\text{LWE}}$[0:3] signals are negated one cycle earlier during writes.

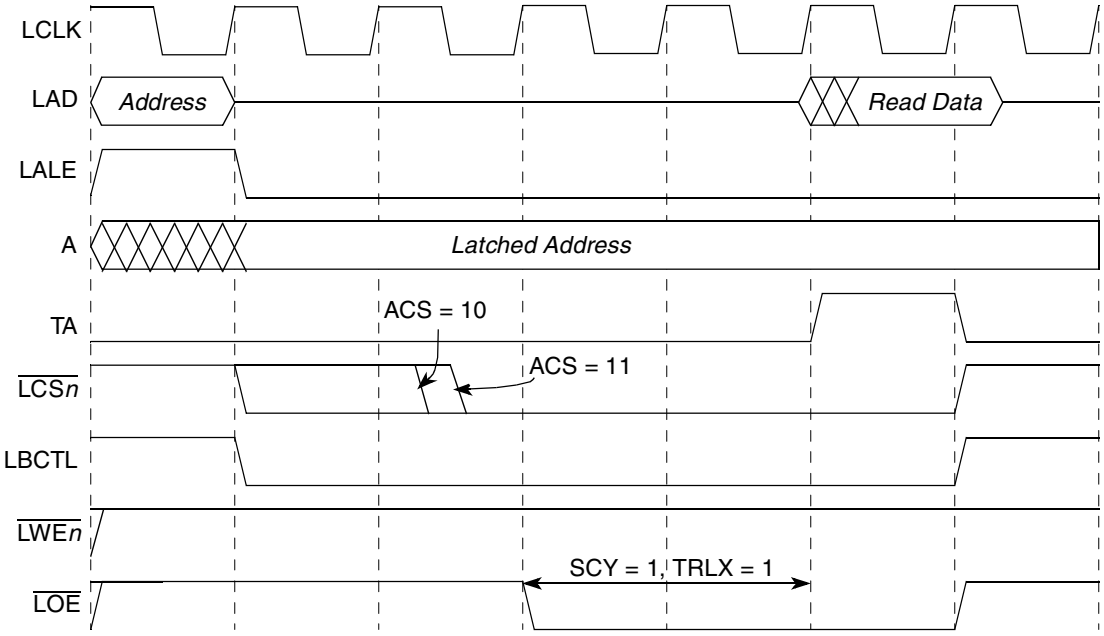


Figure 10-26. GPCM Relaxed Timing Read (XACS = 0, ACS = 1x, SCY = 1, CSNT = 0, TRLX = 1, CLKDIV = 4, 8)

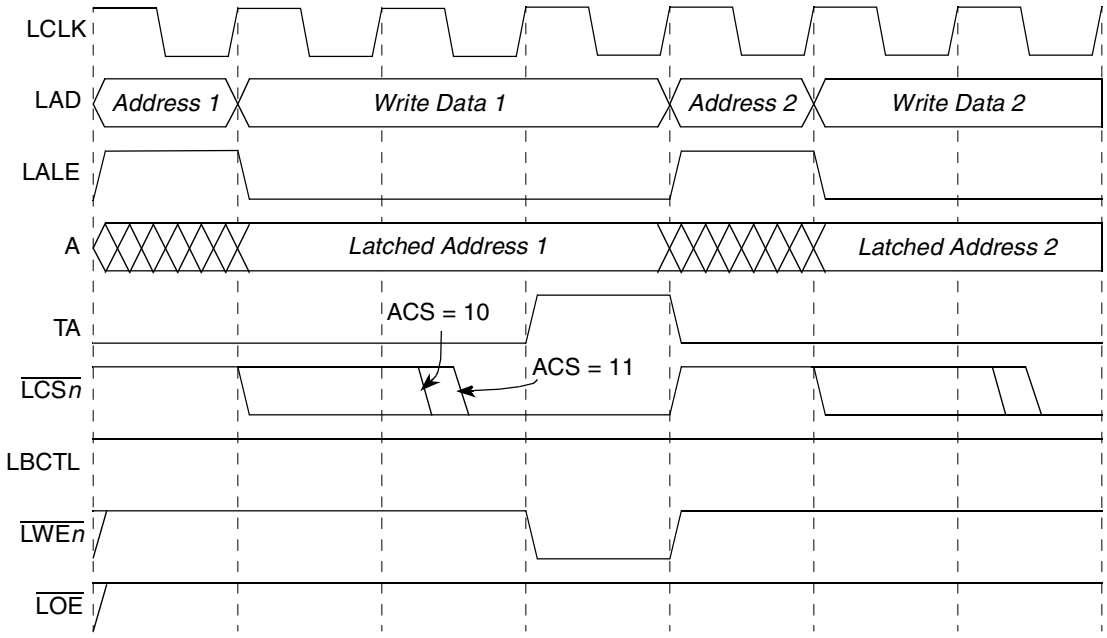


Figure 10-27. GPCM Relaxed Timing Back-to-Back Writes (XACS = 0, ACS = 1x, SCY = 0, CSNT = 0, TRLX = 1, CLKDIV = 4, 8)

Figure 10-26 and Figure 10-27 show relaxed timing read and write transactions. The effect of CLKDIV = 2 for these examples is only to delay the assertion of $\overline{\text{LCS}n}$ in the ACS = 10 case to the ACS = 11 case. The example in Figure 10-27 also shows address and data multiplexing on LAD[0:31] for a pair of writes issued consecutively.

When TRLX and CSNT are set in a write access, the $\overline{\text{LWE}}$[0:3] strobe signals are negated one clock earlier than in the normal case, as shown in Figure 10-28 and Figure 10-29. If ACS ≠ 00, $\overline{\text{LCS}n}$ is also negated one clock earlier.

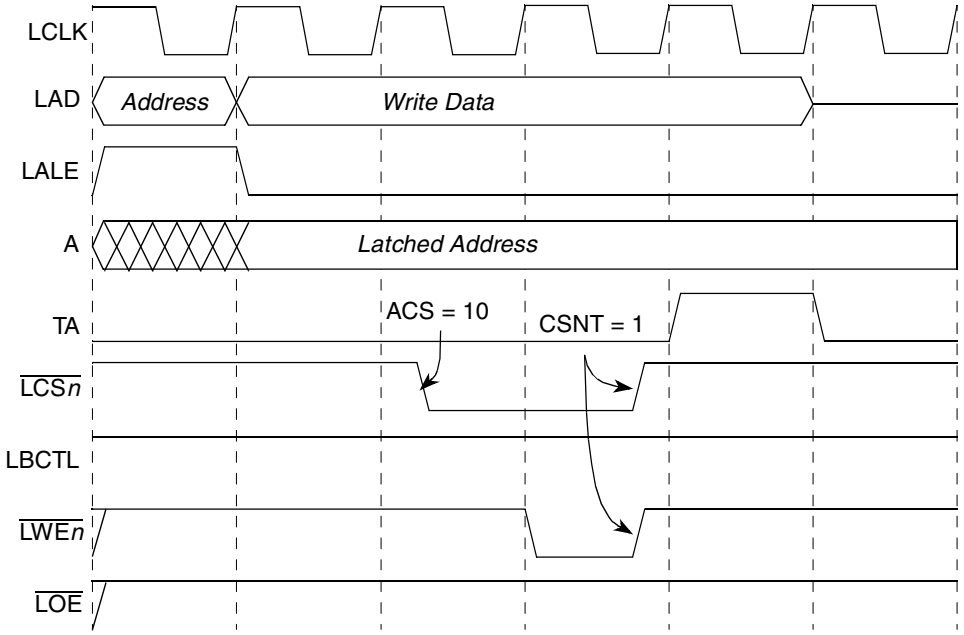


Figure 10-28. GPCM Relaxed Timing Write (XACS = 0, ACS = 10, SCY = 0, CSNT = 1, TRLX = 1, CLKDIV = 4, 8)

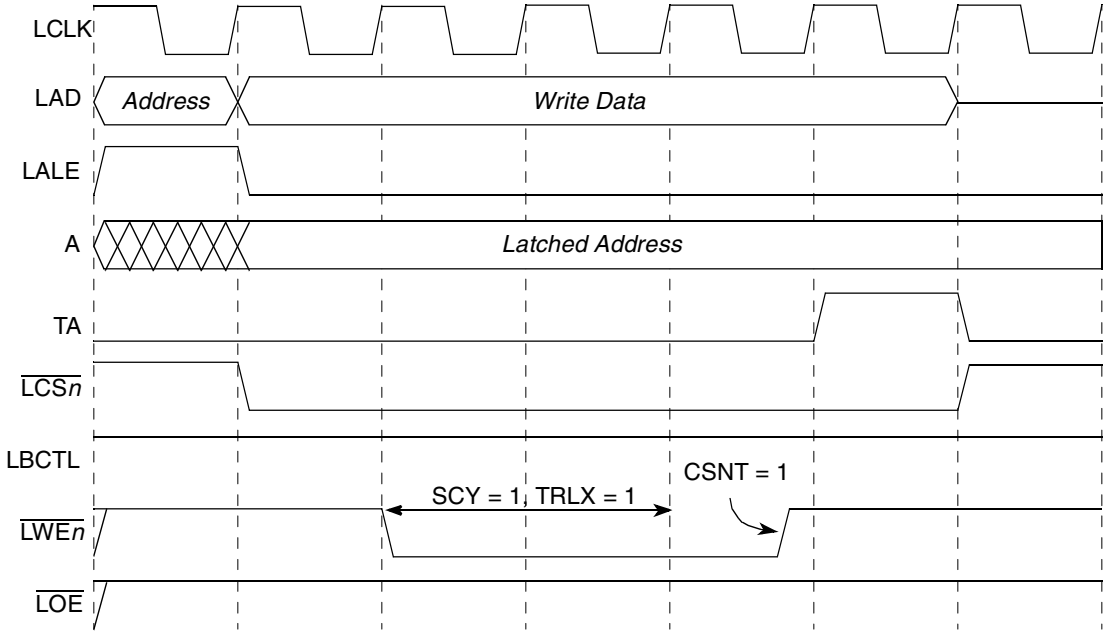


Figure 10-29. GPCM Relaxed Timing Write (XACS = 0, ACS = 00, SCY = 1, CSNT = 1, TRLX = 1, CLKDIV = 4, 8)

10.4.2.2.4 Output Enable ($\overline{\text{LOE}}$) Timing

The timing of the $\overline{\text{LOE}}$ is affected only by 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 $\overline{\text{LCS}n}$ is asserted or coinciding with $\overline{\text{LCS}n}$ (if XACS = 1 and ACS = 10 or ACS = 11). Accordingly, assertion of $\overline{\text{LOE}}$ can be delayed (along with the assertion of $\overline{\text{LCS}n}$) by programming TRLX = 1. $\overline{\text{LOE}}$ negates on the rising clock edge coinciding with $\overline{\text{LCS}n}$ negation

10.4.2.2.5 Extended Hold Time on Read Accesses

Slow memory devices that take a long time to disable their data bus drivers on read accesses should choose some combination of OR*n*[TRLX,EHTR]. Any access following a read access to the slower memory bank is delayed by the number of clock cycles specified in Table 10-6 in addition to any existing bus turnaround cycle. The final bus turnaround cycle is automatically inserted by the LBC for reads, regardless of the setting of OR*n*[EHTR].

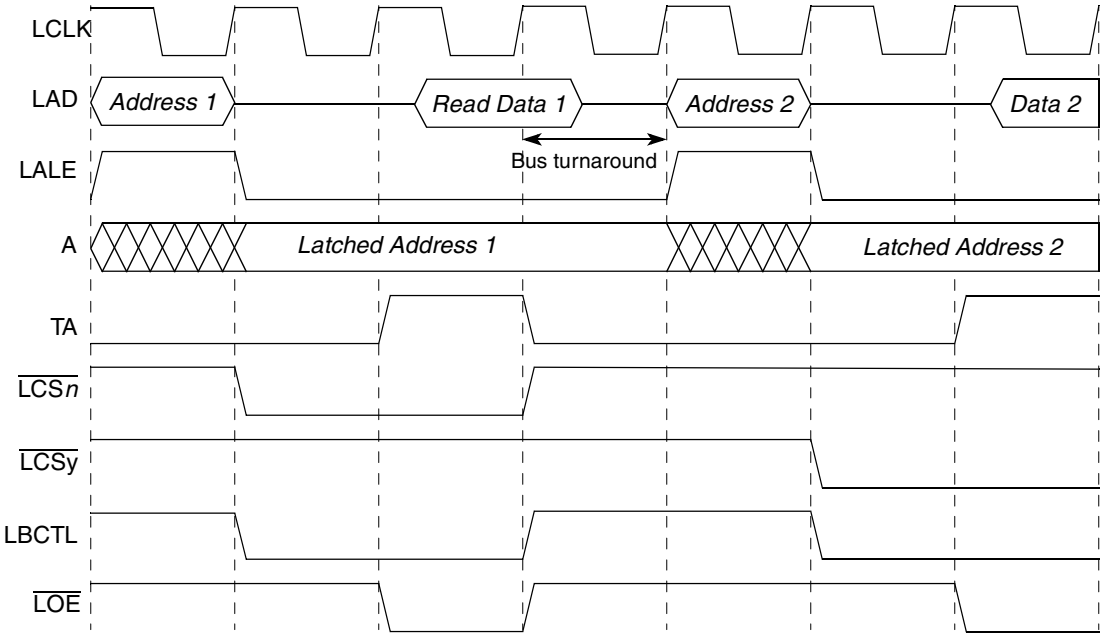


Figure 10-30. GPCM Read Followed by Read (TRLX = 0, EHTR = 0, Fastest Timing)

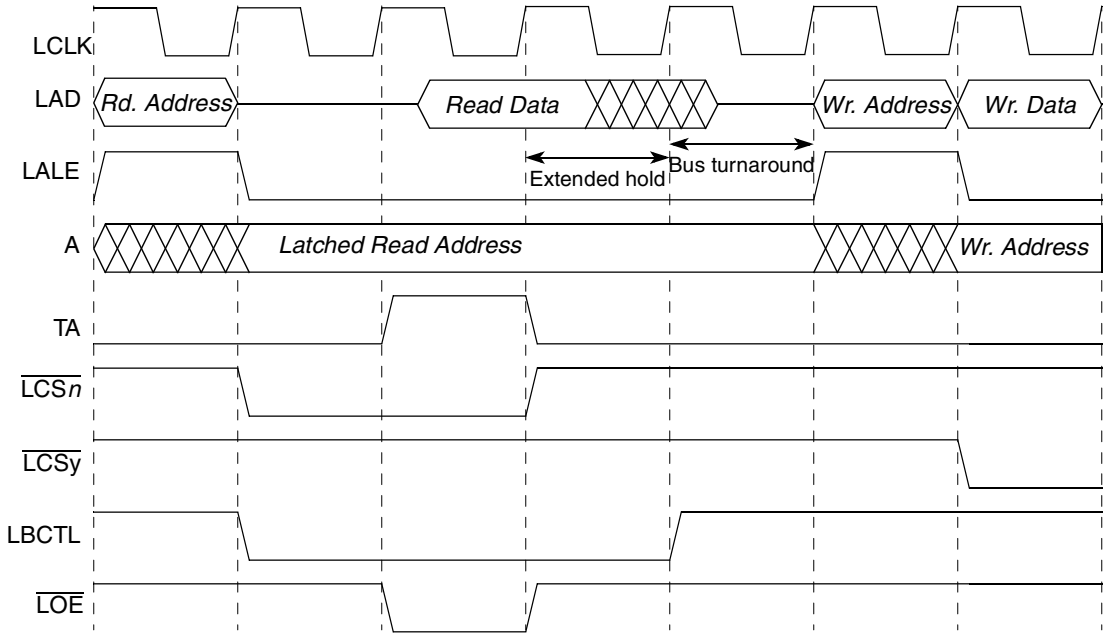


Figure 10-31. GPCM Read Followed by Write (TRLX = 0, EHTR = 1, One-Cycle Extended Hold Time on Reads)

10.4.2.3 External Access Termination ($\overline{\text{LGTA}}$)

External access termination is supported by the GPCM using the asynchronous $\overline{\text{LGTA}}$ input signal, which is synchronized and sampled internally by the local bus. If, during assertion of $\overline{\text{LCS}n}$, the sampled $\overline{\text{LGTA}}$ signal is asserted, it is converted to an internal generation of transfer acknowledge, which terminates the current GPCM access (regardless of the setting of OR*n*[SETA]). $\overline{\text{LGTA}}$ should be asserted for at least one bus cycle to be effective. Note that because $\overline{\text{LGTA}}$ is synchronized, bus termination occurs two cycles after $\overline{\text{LGTA}}$ assertion, so in case of read cycle, the device still must drive data as long as $\overline{\text{LOE}}$ is asserted.

The user selects whether transfer acknowledge is generated internally or externally ($\overline{\text{LGTA}}$) by programming OR*n*[SETA]. Asserting $\overline{\text{LGTA}}$ always terminates an access, even if OR*n*[SETA] = 0 (internal transfer acknowledge generation), but it is the only means by which an access can be terminated if OR*n*[SETA] = 1. The timing of $\overline{\text{LGTA}}$ is illustrated in Figure 10-32.

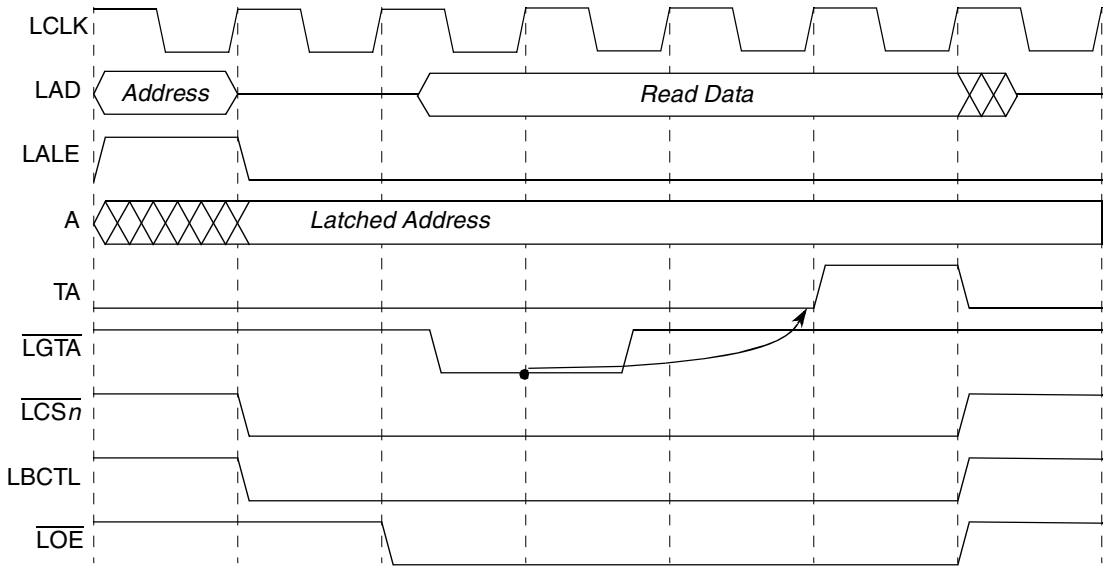


Figure 10-32. External Termination of GPCM Access

10.4.2.4 Boot Chip-Select Operation

Boot chip-select operation allows address decoding for a boot ROM before system initialization. $\overline{\text{LCS0}}$ is the boot chip-select output; its operation differs from other external chip-select outputs after a system reset. When the core begins accessing memory after system reset, $\overline{\text{LCS0}}$ is asserted for every local bus access until BR0 or OR0 is reconfigured.

The boot chip-select also provides a programmable port size, which is configured during reset. The boot chip-select does not provide write protection. $\overline{\text{LCS0}}$ operates this way until the first write to OR0 and it can be used as any other chip-select register after the preferred address range is loaded into BR0. After the first write to OR0, the boot chip-select can be restarted only with a hardware reset. Table 10-27 describes the initial values of the boot bank in the memory controller.

Table 10-27. Boot Bank Field Values After Reset

Register	Field	Setting
BR0	BA PS DECC WP MSEL V	0000_0000_0000_0000_0 From RCWH[ROMLOC] 00 0 000 1

Table 10-27. Boot Bank Field Values After Reset (continued)

Register	Field	Setting
OR0	AM	0000_0000_0000_0000_0
	BCTLD	0
	CSNT	1
	ACS	11
	XACS	1
	SCY	1111
	SETA	0
	TRLX	1
	EHTR	1
	EAD	1

10.4.3 SDRAM Machine

The LBC provides an SDRAM interface (machine) for the local bus. The machine provides the control functions and signals for Intel PC133 and JEDEC-compliant SDRAM devices. Each bank can control an SDRAM device on the local bus.

10.4.3.1 Supported SDRAM Configurations

The memory controller supports any SDRAM configuration with the restrictions that all SDRAM devices that reside on the bus should have the same port size and timing parameters (as defined in LSDMR). Figure 10-33 shows an example connection between the LBC and a 32-bit SDRAM device with 12 address lines. Note that address signals A[2:0] of the SDRAM connect directly to LA[27:29], address signal A10 connects to the LBCs dedicated LSDA10 signal, while the remaining address bits (except A10) are latched from LAD[20:26].

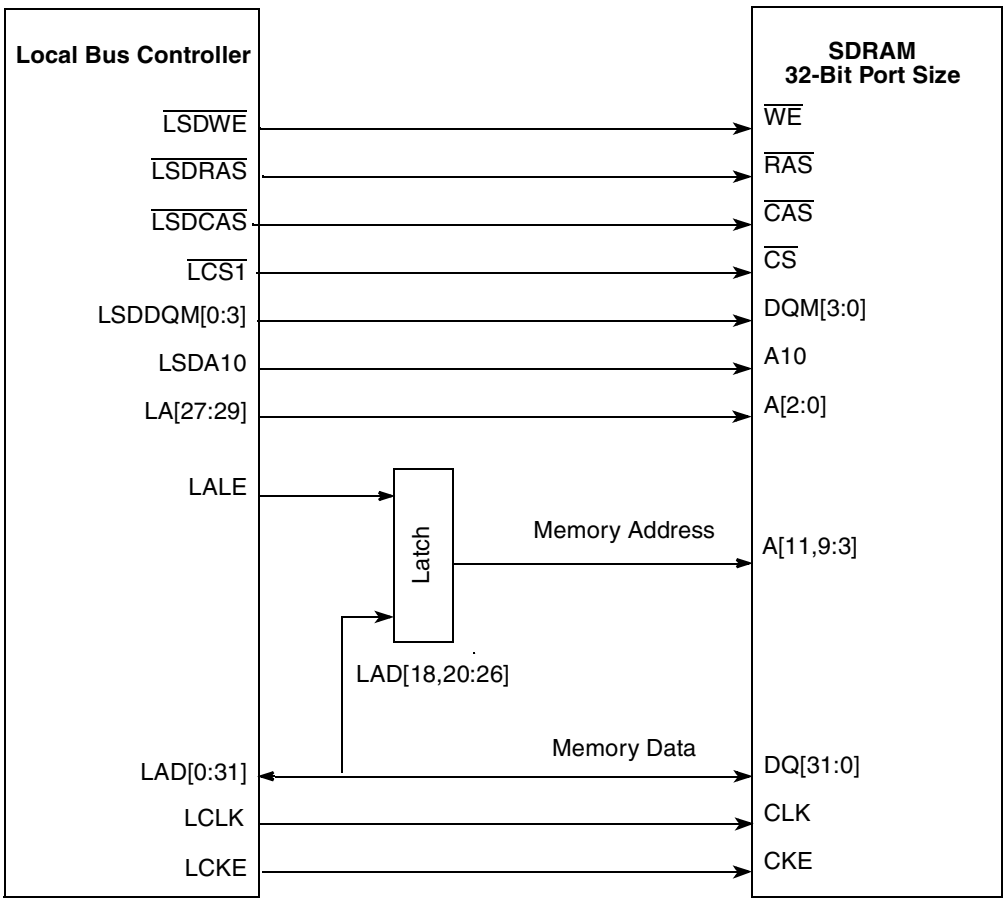


Figure 10-33. Connection to a 32-Bit SDRAM with 12 Address Lines

10.4.3.2 SDRAM Power-On Initialization

Following a system reset, initialization software must set up the programmable parameters in the memory controller banks registers (OR*n*, BR*n*, LSDMR). After all memory parameters are configured, system software should execute the following initialization sequence for each SDRAM device.

- Issue a PRECHARGE-ALL-BANKS command
- Issue eight AUTO-REFRESH commands
- Issue a MODE-SET command to initialize the mode register

The initial commands are executed by setting LSDMR[OP] and accessing the SDRAM with any write that hits the relevant bank. Since the result of any update to the LSDMR must be in effect before accessing the SDRAM with any write, a write to LSDMR should be followed immediately by a read from LSDMR, which must complete prior to an initial write to SDRAM. Further, the first write to SDRAM should be followed immediately by an SDRAM read, which must complete prior to additional LSDMR updates. This enforces a proper ordering between updates to the LSDMR and write accesses to the SDRAM. If the initialization is being done by the e300, this described protocol is guaranteed only if the SDRAM is mapped as cache-inhibited and guarded, as the CCSR memory region containing LSDMR should be. If the

initialization is from an external host, said host must ensure completion of LSDMR and SDRAM reads prior to subsequent writes, as described above.

Note that software should ensure that no memory operations begin until this process completes.

NOTE

In general (not only during power-on reset) the LSDMR/SDRAM access ordering protocol should be observed for proper operation.

10.4.3.3 Intel PC133 and JEDEC-Standard SDRAM Interface Commands

The SDRAM machine performs all accesses to SDRAM by using Intel PC133 and JEDEC-standard SDRAM interface commands. The SDRAM device samples the command and data inputs on the rising edge of the bus clock. Data at the output of the SDRAM device is sampled on the rising edge of the bus clock.

The following SDRAM interface commands are provided by setting LSDMR[OP] to a non-zero value (LSDMR[OP] = 000 sets normal read/write operation):

Table 10-28. SDRAM Interface Commands

Command (LSDMR[OP])	Description
ACTIVATE (110)	Latches the row address and initiates a memory read of that row. Row data is latched in SDRAM sense amplifiers and must be restored with a PRECHARGE command before another ACTIVATE is issued.
MODE-SET (011)	Allows setting of SDRAM options—CAS latency and burst length. CAS latency depends on the SDRAM device used. Although some SDRAMs provide burst lengths of 1, 2, 4, 8, or a page, the local bus memory controller supports only 8-beat bursts for 8-bit and 32-bit port size, or 4-beat bursts for 16-bit port size. The LBC does not support burst lengths of 1, 2 and a page for SDRAMs. The mode register data (CAS latency and burst length) is programmed into the LSDMR register by initialization software after reset. After the LSDMR is set, the LBC transfers the information to the SDRAM device by issuing a MODE-SET command.
PRECHARGE (100: single bank) (101: all-banks)	Restores data from the sense amplifiers to the appropriate row in the SDRAM device array. Also initializes the sense amplifiers to prepare for activating another row in the SDRAM device. Note that the LBC uses LSDA10 to distinguish between PRECHARGE-ALL-BANKS (LSDA10 is high) and PRECHARGE-SINGLE-BANK (LSDA10 is low). The SDRAMs must be compatible with this format.
READ (111)	Latches the column address and transfers data from the selected sense amplifier on the SDRAM device, to the output buffer as determined by the column address. During each successive clock, additional data is driven without additional read commands. At the end of the burst, the page remains open. Burst length is the one set for this bank. Read data is discarded by the LBC.
WRITE (111)	Latches the column address and transfers data from the data signals to the selected sense amplifier on the SDRAM device, as determined by the column address. During each successive clock, additional data is transferred to the sense amplifiers from the data signals without additional write commands. At the end of the burst, the page remains open. Burst length is the one set for this bank. LSDDQM[0:3] are inactive and write data is undefined.

Table 10-28. SDRAM Interface Commands (continued)

Command (LSDMR[OP])	Description
AUTO-REFRESH (001)	Causes a row to be read in all memory banks (JEDEC SDRAM) as determined by the refresh row address counter (similar to CBR). The refresh row address counter is internal to the SDRAM device. After being read, a row is automatically rewritten into the memory array. All banks must be in a precharged state before executing refresh.
SELF-REFRESH (010)	Allows data to be retained in the SDRAM device, even without any active clocks. When placed in this mode, the SDRAM device can issue its own refresh commands, without external clocking from the LBC and the LCKE signal from the LBC is negated. This command can be issued at any time. Normal operation can be resumed only by setting LSDMR[OP] = 000, and waiting a at least of 200 bus cycles before issuing reads or writes to the LBC.

10.4.3.4 Page Hit Checking

The SDRAM machine supports page-mode operation. Each time a page is activated on the SDRAM device, the SDRAM machine stores its address in a page register. The page information, which the user writes to the OR*n* register, is used along with the bank size to compare page bits of the address to the page register each time a bus-cycle access is requested. If a match is found, together with a bank match, the bus cycle is defined as a page hit. An open page is automatically closed by the SDRAM machine if the bus becomes idle, unless OR*n*[PMSEL] = 1.

10.4.3.5 Page Management

The LBC can manage at most four open pages (one page per SDRAM bank) for a single SDRAM device. After a page is opened, it remains open unless one of the following occurs:

- The next access is to a page in a different SDRAM device, in which case all open pages on the current device are closed with a PRECHARGE-ALL-BANKS command.
- The next access is to a page in an SDRAM bank that has a different page open on it, in which case the old page is closed with a PRECHARGE-SINGLE-BANK command.
- The current SDRAM device requires refresh services, in which case all open pages on the current device are closed with a PRECHARGE-ALL-BANKS command.
- The bus becomes idle and OR*n*[PMSEL] = 0, in which case all open pages in the current device are closed with a PRECHARGE-ALL-BANKS command.

10.4.3.6 SDRAM Address Multiplexing

The lower address bus bits are connected to the memory device's address port with the memory controller multiplexing the row/column and the internal bank select lines. The position of the bank select lines are set according to LSDMR[BSMA]. Figure 10-34 shows how the SDRAM controller shifts the row address down to the lower output address signals during activate and shifts the bank select bits up to the address signals specified by LSDMR[BSMA], supporting page-based interleaving. The lsb of the logical row address (A*n* in Figure 10-34) is aligned with the connected lsb of LAD (bits 29, 30, and 31 for port sizes of 32, 16, and 8 bits, respectively).

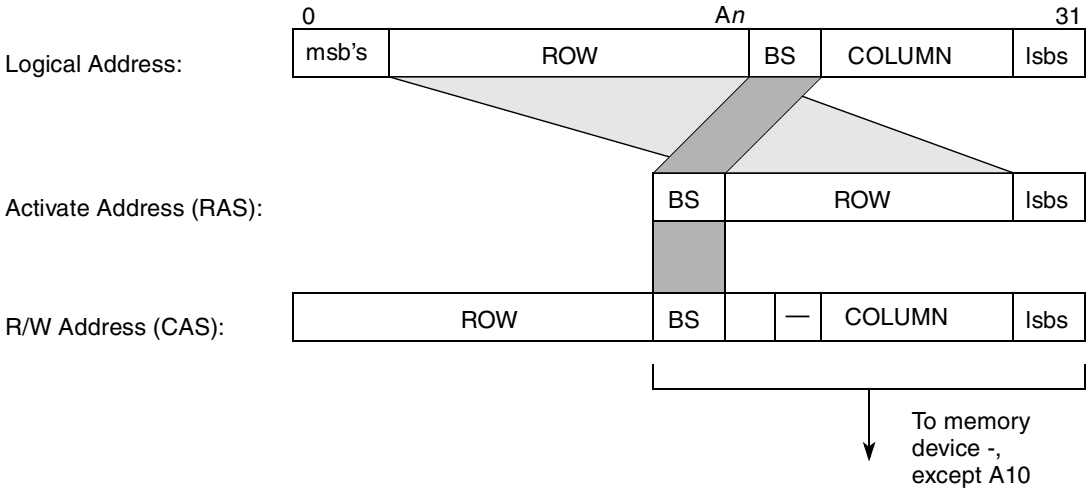


Figure 10-34. SDRAM Address Multiplexing

Note that during normal operation (read/write), a full 32-bit address that includes row and column is generated on LAD[0:31]. However, address/data signal multiplexing implies that the address must be latched by an external latch that is controlled by LALE. All SDRAM device address signals need to be connected to the latched address bits and burst address bits (LA[27:31]) of the LBC, with the exception of A10, which has a dedicated connection on LSDA10. LSDA10 is driven with the appropriate row address bit for SDRAM commands that require A10 to be an address.

10.4.3.7 SDRAM Device-Specific Parameters

The software is responsible for setting correct values for device-specific parameters that can be extracted from the device's data sheet. The values are stored in the OR*n* and LSDMR registers. These parameters include the following:

- Precharge to activate interval (LSDMR[PRETOACT])
- Activate to read/write interval (LSDMR[ACTTORW])
- CAS latency, column address to first data out (LSDMR[CL] and LCRR[ECL])
- Write recovery, last data in to precharge (LSDMR[WRC])
- Refresh recovery interval (LSDMR[RFRC])
- External buffers on the control lines present (LSDMR[BUFCMD] and LCRR[BUFCMDC])

In addition, the LBC hardware ensures a default activate to precharge interval of 10 bus cycles. The following sections describe SDRAM parameters programmed in LSDMR.

10.4.3.7.1 Precharge-to-Activate Interval

The precharge-to-activate interval parameter, controlled by LSDMR[PRETOACT], defines the earliest timing for an ACTIVATE or REFRESH command after a PRECHARGE command to the same SDRAM bank.

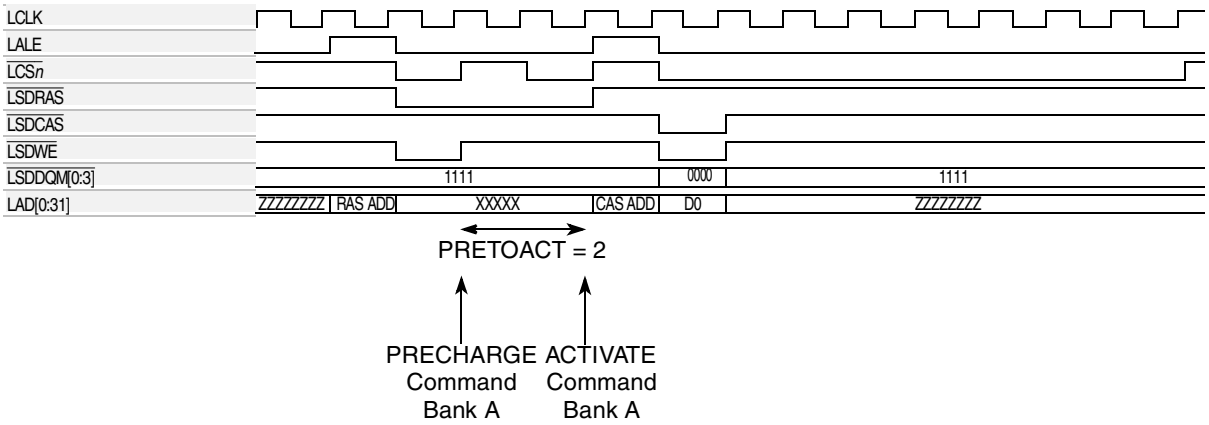


Figure 10-35. PRETOACT = 2 (2 Clock Cycles)

10.4.3.7.2 Activate-to-Read/Write Interval

This active-to-read/write interval parameter, controlled by LSDMR[ACTTORW], defines the earliest timing for a READ/WRITE command after an ACTIVATE command to the same SDRAM bank.

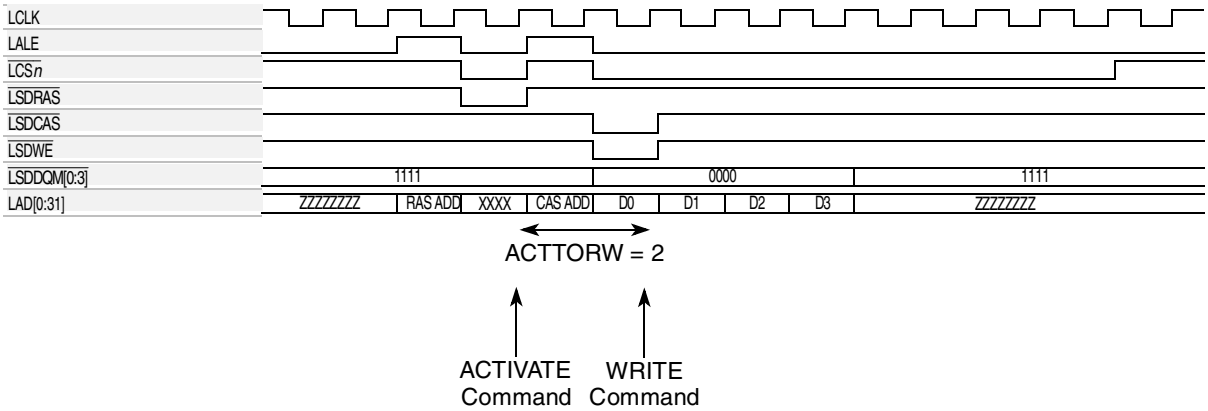


Figure 10-36. ACTTORW = 2 (2 Clock Cycles)

10.4.3.7.3 Column Address to First Data Out—CAS Latency

This parameter, controlled by LSDMR[CL] for latency of 1, 2, or 3 and by LCRR[ECL] for latency of more than 3, defines the timing for first read data after a column address is sampled by the SDRAM.

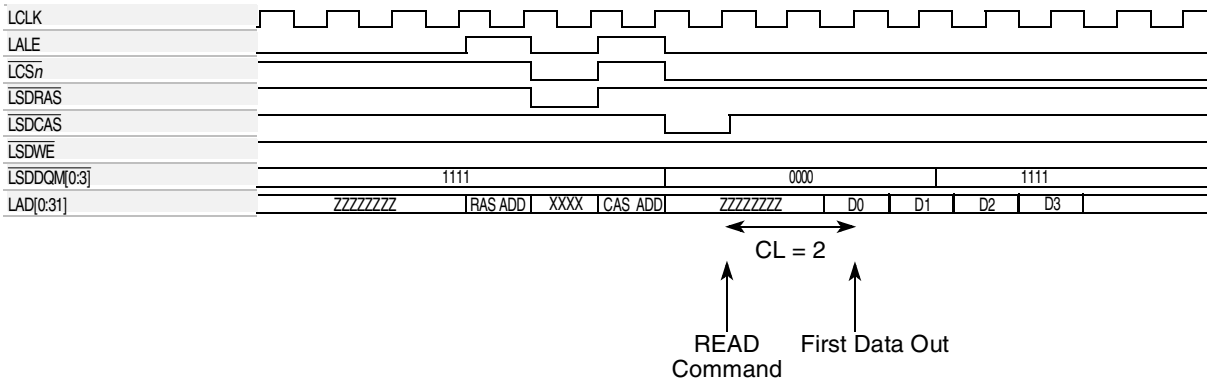


Figure 10-37. CL = 2 (2 Clock Cycles)

10.4.3.7.4 Last Data In to Precharge—Write Recovery

This parameter, controlled by LSDMR[WRC], defines the earliest timing for a PRECHARGE command after the last data was written to the SDRAM.

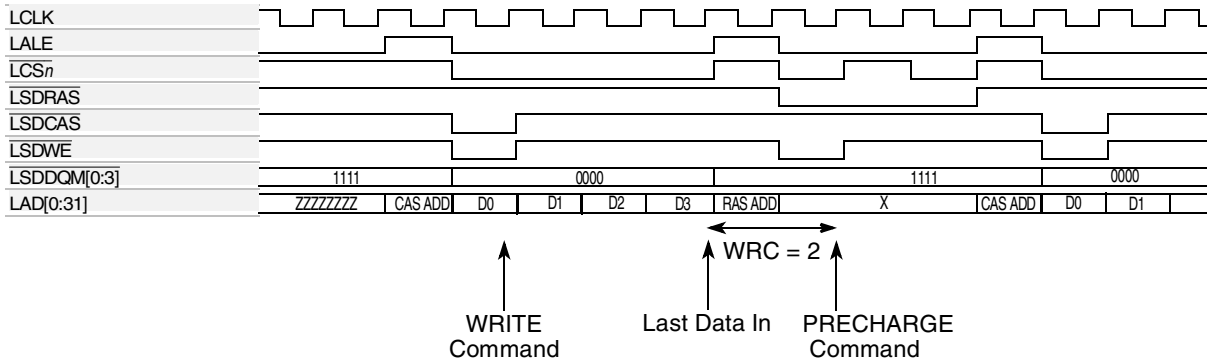


Figure 10-38. WRC = 2 (2 Clock Cycles)

10.4.3.7.5 Refresh Recovery Interval (RFRC)

This parameter, controlled by LSDMR[RFRC], defines the earliest timing for an ACTIVATE or REFRESH command after a REFRESH command to the same SDRAM device.

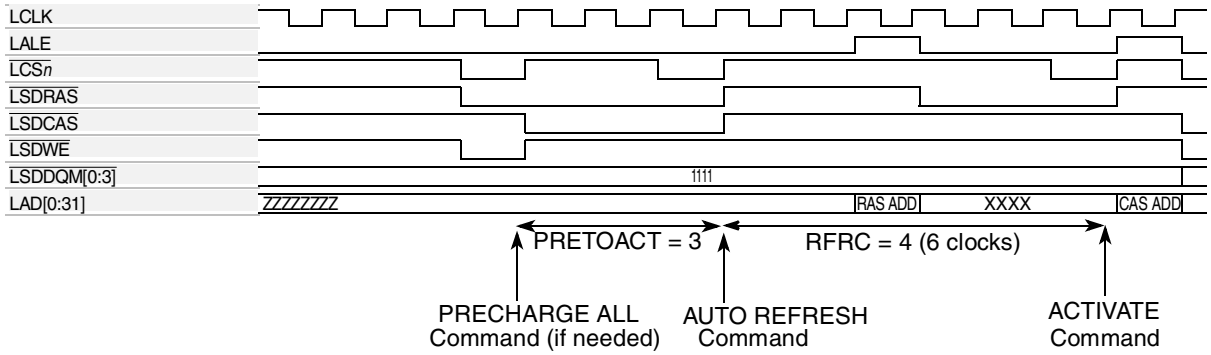


Figure 10-39. RFRC = 4 (6 Clock Cycles)

10.4.3.7.6 External Address and Command Buffers (BUFCMD)

If the additional delay of any buffers placed on the command strobes ($\overline{\text{LSDRAS}}$, $\overline{\text{LSDCAS}}$, $\overline{\text{LSDWE}}$ and LSDA10), is endangering the device setup time, LSDMR[BUFCMD] should be set. Setting this bit causes the memory controller to add LCRR[BUFCMDC] extra bus cycles to the assertion of SDRAM control signals ($\overline{\text{LSDRAS}}$, $\overline{\text{LSDCAS}}$, $\overline{\text{LSDWE}}$ and LSDA10) for each SDRAM command.

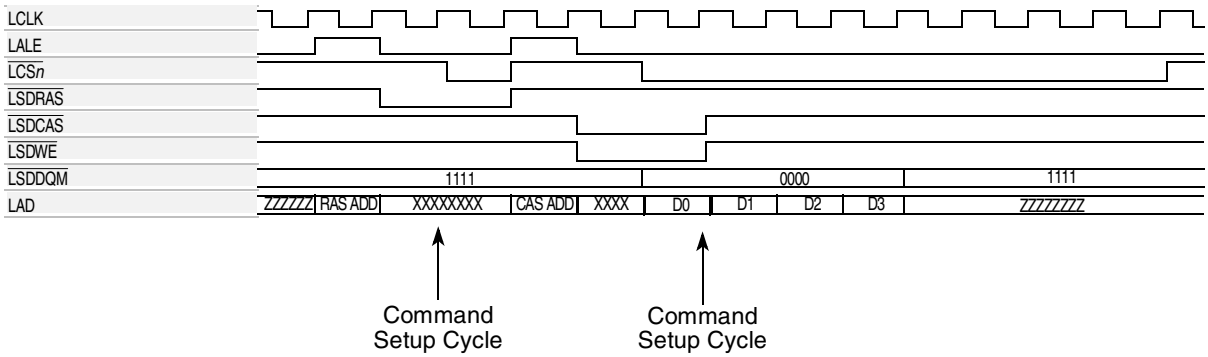


Figure 10-40. BUFCMD = 1, LCRR[BUFCMDC] = 2

10.4.3.8 SDRAM Interface Timing

The following figures show SDRAM timing for various types of accesses.

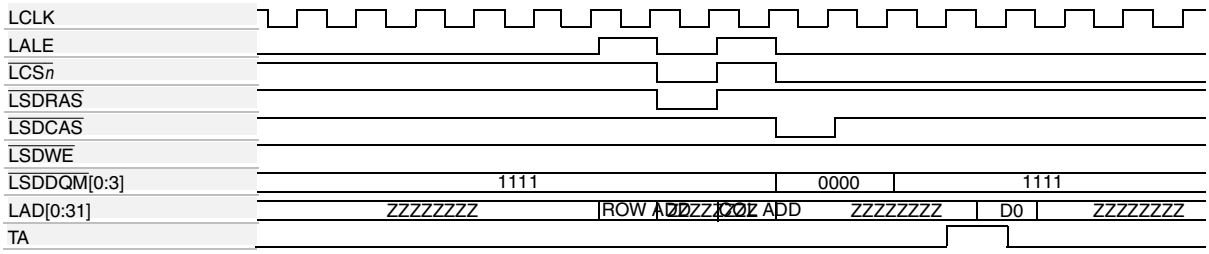


Figure 10-41. SDRAM Single-Beat Read, Page Closed, CL = 3

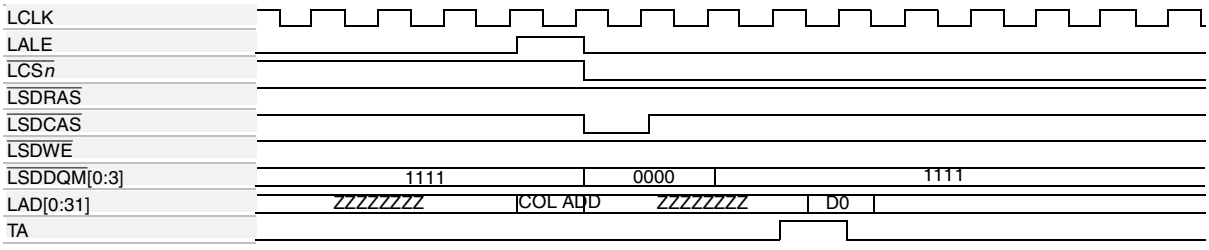


Figure 10-42. SDRAM Single-Beat Read, Page Hit, CL = 3

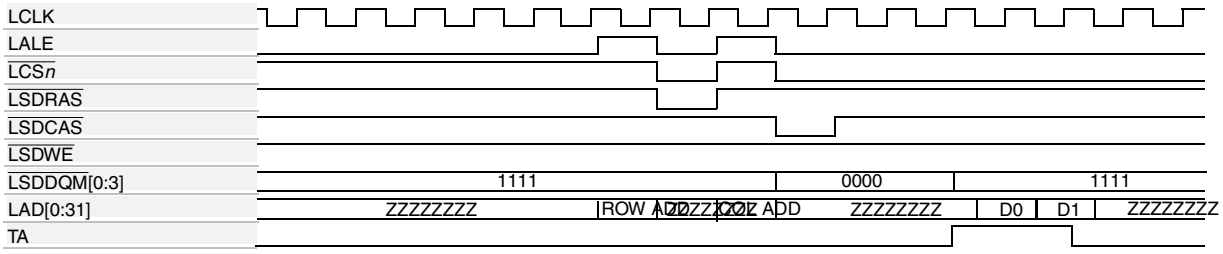


Figure 10-43. SDRAM Two-Beat Burst Read, Page Closed, CL = 3

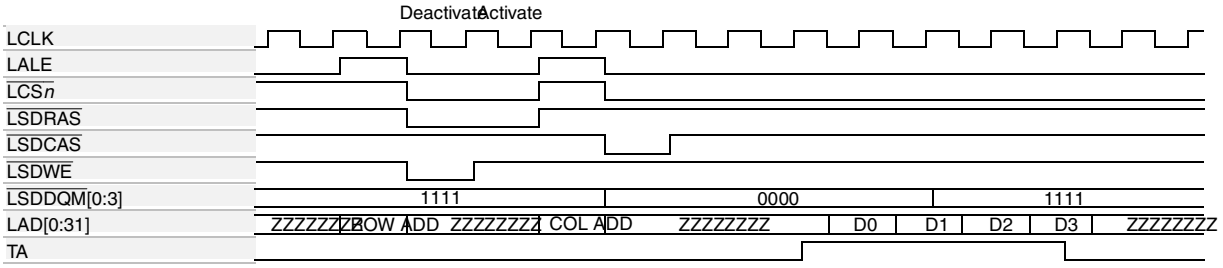


Figure 10-44. SDRAM Four-Beat Burst Read, Page Miss, CL = 3

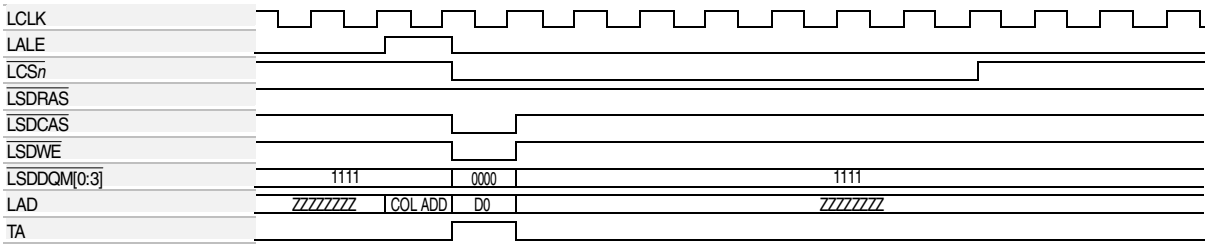


Figure 10-45. SDRAM Single-Beat Write, Page Hit

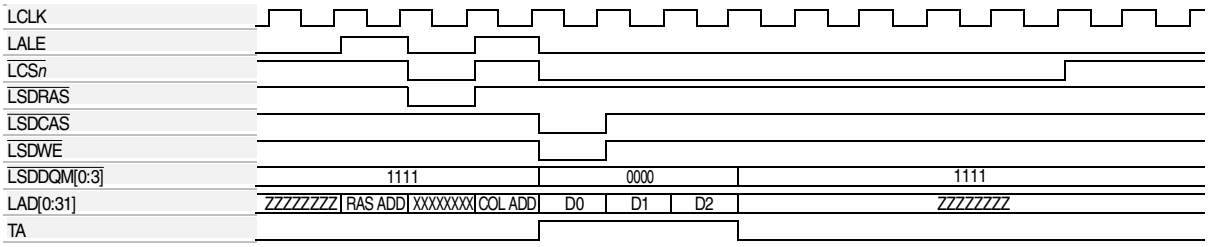


Figure 10-46. SDRAM Three-Beat Write, Page Closed

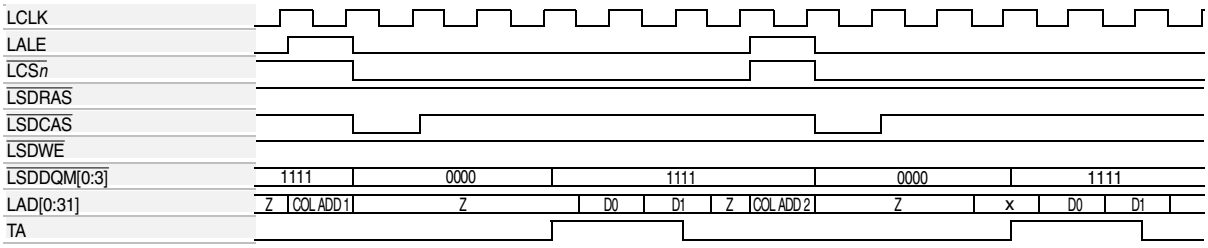


Figure 10-47. SDRAM Read-after-Read Pipelined, Page Hit, CL = 3

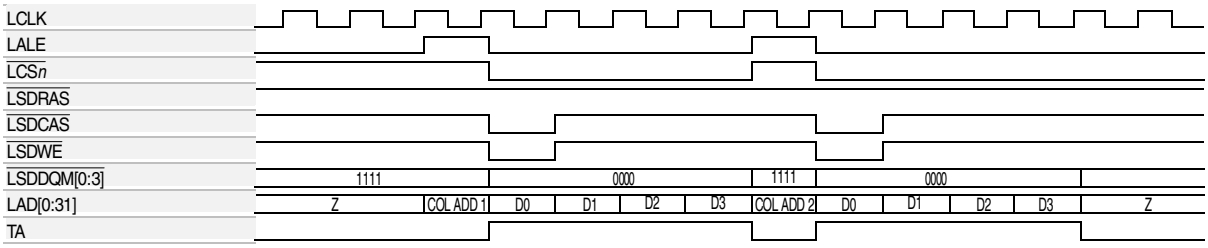


Figure 10-48. SDRAM Write-after-Write Pipelined, Page Hit

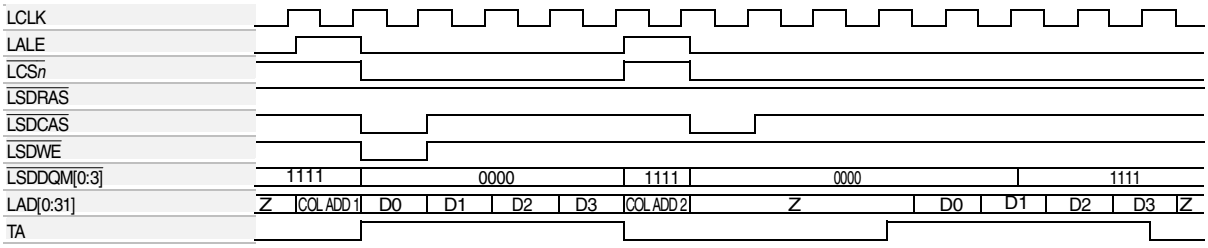


Figure 10-49. SDRAM Read-after-Write Pipelined, Page Hit

10.4.3.9 SDRAM Read/Write Transactions

The SDRAM interface supports read and write transactions of between 1 and 8 data beats for transaction sizes ranging from 1 to 32 bytes. A full burst is performed for each transaction, with the burst length dependent on the port size. A maximum burst of 8 beats is used for an 8- or 32-bit port size, while a maximum burst of 4 beats is used for a 16-bit port size, as programmed in LSDMR[BL]. For reads that require less than the full burst length, extraneous data in the burst is ignored and suppressed by the assertion of LSDDQM[0:3]. For writes that require less than the full burst length, the non-targeted addresses are protected by driving corresponding LSDDQM bits high (inactive) on the irrelevant cycles of the burst. However, system performance is not compromised because, if a new transaction is pending, the SDRAM controller begins executing it immediately, effectively terminating the burst early.

10.4.3.10 SDRAM MODE-SET Command Timing

The LBC transfers mode register data (CAS latency and burst length) stored in the LSDMR register to the SDRAM device by issuing the MODE-SET command, as shown in Figure 10-50. In this case, the latched address carries the mode bits for the command.

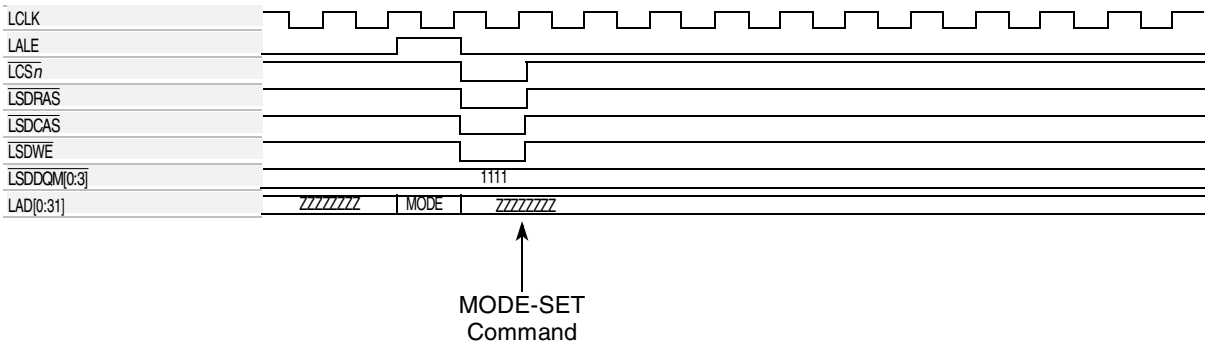


Figure 10-50. SDRAM MODE-SET Command

10.4.3.11 SDRAM Refresh

The memory controller supplies AUTO-REFRESH commands to any connected SDRAM device according to the interval specified in LSRT (and prescaled by MRTPR[PTP]). This interval represents the time period required between refreshes. The values of LSRT and MRTPR depend on the specific SDRAM devices used and the system clock frequency of the LBC. This value should allow for a potential collision between memory accesses and refresh cycles. The refresh interval period must be greater than the access time. To ensure that read and write operations complete successfully.

There are two levels of refresh request priority—low and high. The low priority request is generated as soon as the refresh timer expires; this request is granted only if no other requests to the memory controller are pending. If the request is not granted (memory controller is busy) and the refresh timer expires two more times, the request becomes high priority and is served when the current memory controller operation finishes.

10.4.3.11.1 SDRAM Refresh Timing

The SDRAM memory controller implements bank staggering for the auto refresh function. This reduces instantaneous current consumption for memory refresh operations.

After a refresh request is granted, the memory controller begins issuing an AUTO-REFRESH command to each device associated with the refresh timer. After a refresh command is issued to an SDRAM device, the memory controller waits for the number of bus clock cycles programmed in the SDRAM machine's mode register (LSDMR[RFRC]) before issuing any subsequent ACTIVATE command to the same device. To avoid violating SDRAM device timing constraints, the user should ensure that the refresh request interval, defined by LSRT and MRTPR, is greater than the refresh recovery interval, defined by LSDMR[RFRC].

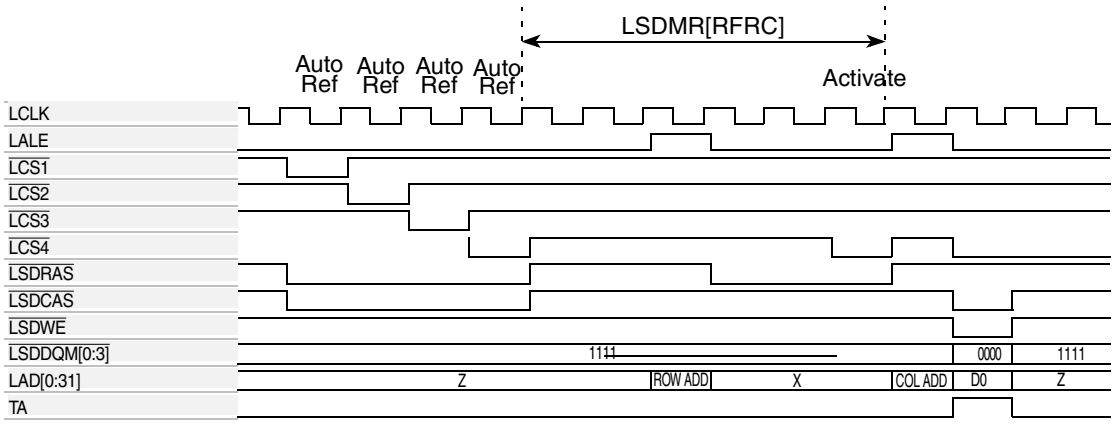


Figure 10-51. SDRAM Bank-Staggered Auto Refresh Timing

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 ($\overline{\text{LCS}}n$, $\overline{\text{LBS}}$[0:3] and $\overline{\text{LGPL}}$[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. Figure 10-52 shows the basic operation of each UPM.

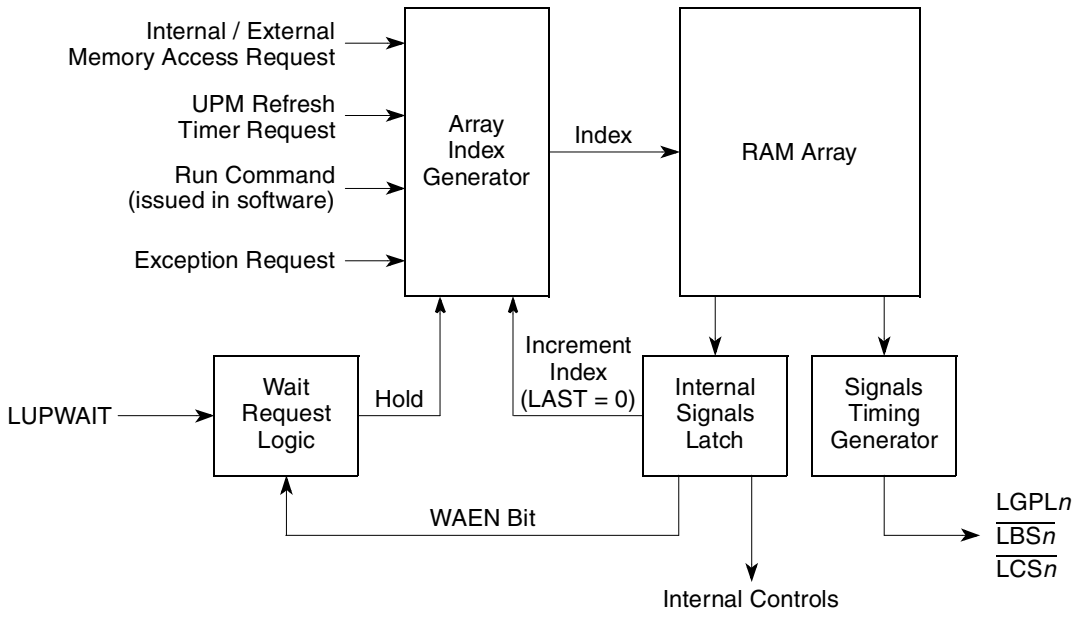


Figure 10-52. User-Programmable Machine Functional Block Diagram

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 possible UPM request. An internal device's request for a memory access initiates one of the following patterns (M*n*MR[OP] = 00):

- Read single-beat pattern (RSS)
- Read burst cycle pattern (RBS)
- Write single-beat pattern (WSS)
- Write burst cycle pattern (WBS)

A UPM refresh timer request pattern initiates a refresh timer pattern (RTS).

An exception (caused by a bus monitor time-out error) occurring while another UPM pattern is running initiates an exception condition pattern (EXS).

Figure 10-53 and Table 10-29 show the start addresses of these patterns in the UPM RAM, according to cycle type. RUN commands (M*n*MR[OP] = 11), however, can initiate patterns starting at any of the 64 UPM RAM words.

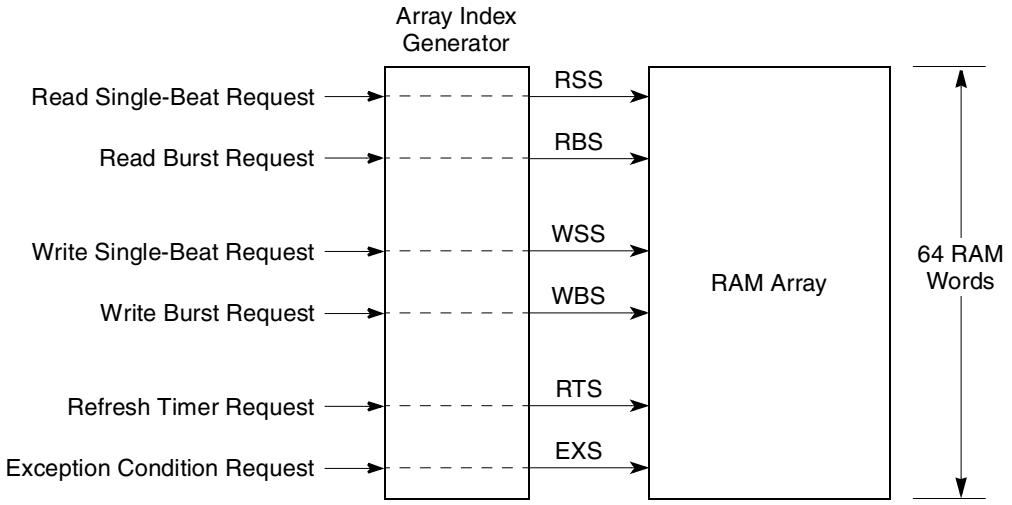


Figure 10-53. RAM Array Indexing

Table 10-29. UPM Routines Start Addresses

UPM Routine	Routine Start Address
Read single-beat (RSS)	0x00
Read burst (RBS)	0x08
Write single-beat (WSS)	0x18
Write burst (WBS)	0x20
Refresh timer (RTS)	0x30
Exception condition (EXS)	0x3C

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 four double words regardless of port size. For 32-bit accesses, the burst cycle starts with one transfer start but ends after eight transfer acknowledges, whereas an 8-bit device requires 32 transfer acknowledges.

The user must ensure that patterns for single-beat transfers contain one, and only one, transfer acknowledge (UTA bit in RAM word set high) and for a burst transfer, contain the exact number of transfer acknowledges required.

Any transfers that do not naturally fit single or burst transfers are synthesized as a series of single transfers. These accesses are treated by the UPM as back-to-back, single-beat transfers. Burst transfers can also be inhibited by setting OR*n*[BI]. Burst performance can be achieved by ensuring that UPM transactions are 32-byte aligned with a transaction size being some multiple of 32-bytes, which is a natural fit for cache-line transfers, for example.

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 10-54 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.

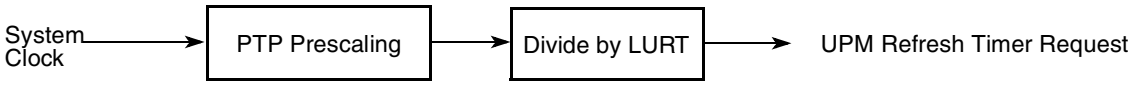


Figure 10-54. Memory Refresh Timer Request Block Diagram

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 refresh pattern if the RFEN bit of the corresponding UPM is set. 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 M*n*MR[RFEN] bits are set.

Note that the UPM refresh timer request should not be used in a system with SDRAM refresh enabled. The system designer must choose to use either SDRAM refresh or UPM refresh. Using both may result in missing refresh periods to memory.

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. Other memory devices require special commands to be issued on their control signals, such as for SDRAM initialization.

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*n*MR[OP] = 11 and accessing UPM*n* memory region with any write transaction that hits the corresponding UPM machine. M*n*MR[MAD] determines the starting address in the RAM array for the pattern.

Note that transfer acknowledges (UTA bit in the RAM word) are ignored for software (RUN command) requests, and hence the LAD signals remain high-impedance unless the normal initial LALE occurs or the RUN pattern causes assertion of LALE to occur on changes to the RAM word AMX field.

10.4.4.1.4 Exception Requests

When the LBC 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*n*MR.

Patterns are written to the RAM array by setting M*n*MR[OP] = 01 and accessing the UPM with any write transaction that hits the relevant chip select. The entire array is thus programmed by an alternating series of writes: to MDR (RAM word to be written) each time followed by a read from MDR and then followed by a (dummy) write transaction to the relevant UPM assigned bank. A read from MDR is required to ensure that the MDR update has occurred prior to the (dummy) write transaction.

RAM array contents may also be read for debug purposes, for example, by alternating dummy read transactions, each time followed by reads of MDR (when M*n*MR[OP] = 10).

M*n*MR / MDR registers should not be updated while dummy read/write access is still in progress. If the M*n*MR[MAD] is incremented, the previous dummy transaction is already completed. To enforce proper ordering between updates to the M*n*MR/MDR register and the dummy accesses to the UPM memory region, two rules must be followed:

- Because the result of any update to the M*n*MR/MDR register must be in effect before the dummy read or write to the UPM region, a write to M*n*MR/MDR should be followed immediately by a read of M*n*MR/MDR.
- The UPM memory region should have the same MMU settings as the memory region containing the M*n*MR configuration register; both should be mapped by the MMU as Cache-Inhibited and Guarded. This prevents the e300c1 core from reordering a read of the UPM memory around the read of M*n*MR. When 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.

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*x* and OR*x* registers have been previously set up:

1. Program M*n*MR for the first write (with the desired RAM array address).
2. Write pattern/data to MDR to ensure that the M*n*MR 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 M*n*MR register if step 2 is not performed.)
4. Perform a dummy write transaction. (Write transaction can now be performed.)
5. Read/check M*n*MR[MAD]. If incremented, the previous dummy write transaction is completed; proceed to step 6. Repeat step 5 until incremented.
6. Program M*n*MR for the second write with the desired RAM array address.
7. Write pattern/data to MDR to ensure that the M*n*MR has already been updated with the desired configuration.
8. Read MDR to ensure that the MDR has already been updated with the desired pattern.
9. Perform a dummy write transaction. (Write transaction can now be performed.)
10. Read/check M*n*MR[MAD]. If incremented, the previous dummy write transaction is completed.

Note that if step 1 (or 6) and 2 (or 7) are reversed, step 3 (or 8) is replaced by the following:

- Read M*n*MR to ensure that the M*n*MR 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 (M*n*MR[OP] = 0b10). The following example further illustrates the steps required to perform two reads from the RAM array at non-sequential addresses assuming that the relevant BR*x* and OR*x* registers have been previously set up:

1. Program M*n*MR for the first read with the desired RAM array address.
2. Read M*n*MR to ensure that the M*n*MR has already been updated with the desired configuration, such as RAM array address.
3. Perform a dummy read transaction. (Read transaction can now be performed.)
4. Read/check M*n*MR[MAD]. If incremented, the previous dummy read transaction is completed; proceed to step 5. Repeat step 4 until incremented.
5. Read MDR.
6. Program M*n*MR for the second read with the desired RAM array address.
7. Read M*n*MR to ensure that the M*n*MR has already been updated with the desired configuration, such as RAM array address.
8. Perform a dummy read transaction. (Read transaction can now be performed.)
9. Read/check M*n*MR[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 $\overline{\text{LCS}n}$ and $\overline{\text{LBS}}$ 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 10-55 and Figure 10-56. 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.

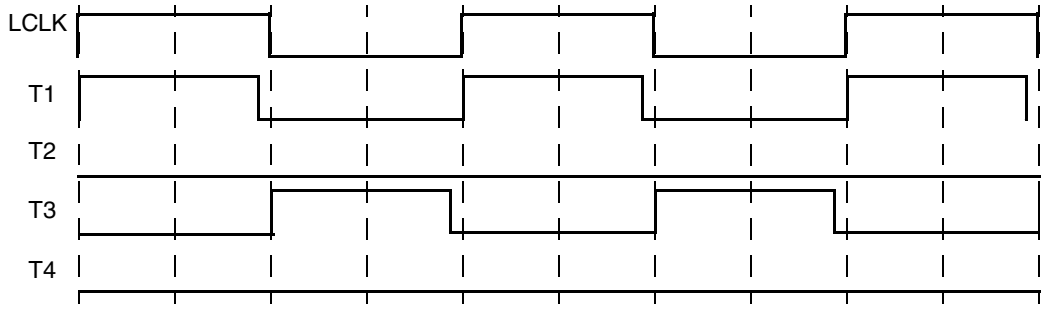


Figure 10-55. UPM Clock Scheme for LCRR[CLKDIV] = 2

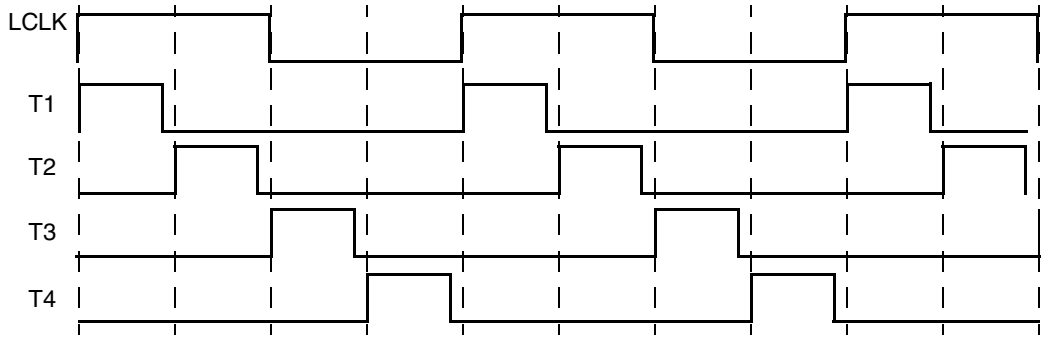


Figure 10-56. UPM Clock Scheme for LCRR[CLKDIV] = 4 or 8

10.4.4.4 UPM RAM Array

The RAM array for each UPM is 64 locations deep and 32 bits wide, as shown in Figure 10-57. The signals at the bottom of the figure are UPM outputs. The selected $\overline{\text{LCS}n}$ is for the bank that matches the current address. The selected $\overline{\text{LBS}}$ is for the byte lanes read or written by the access.

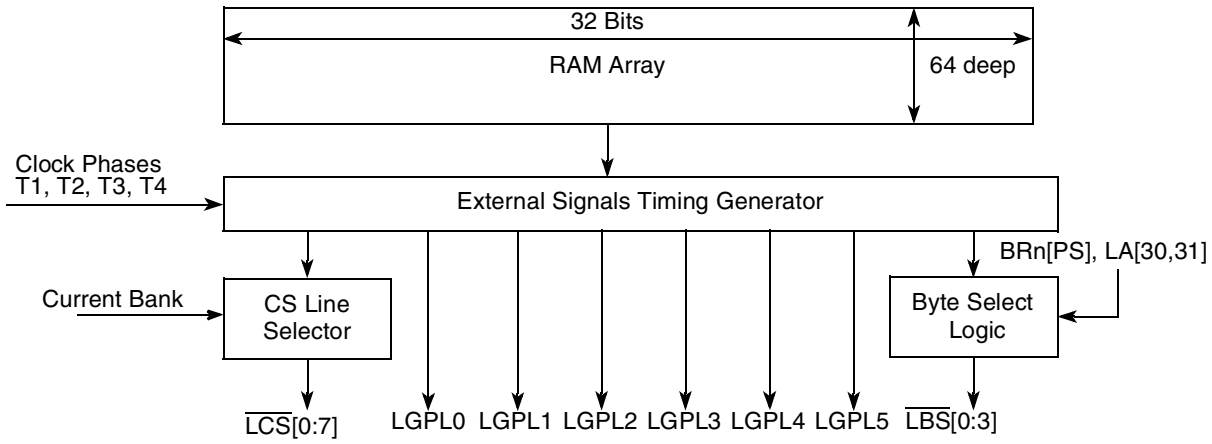


Figure 10-57. UPM RAM Array and Signal Generation

10.4.4.4.1 UPM 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 10-58 shows the RAM word fields. When LCRR[CLKDIV] = 4 or 8, the CST*n* and BST*n* bits determine the state of UPM signals $\overline{\text{LCS}n}$ and $\overline{\text{LBS}}$[0:3] at each quarter phase of the bus clock. When LCRR[CLKDIV] = 2, CST2 and CST4 are ignored and the external has the values defined by CST1 and CST3 but extended to half the clock cycle in duration. The same interpretation occurs for the BST*n* bits when LCRR[CLKDIV] = 2.

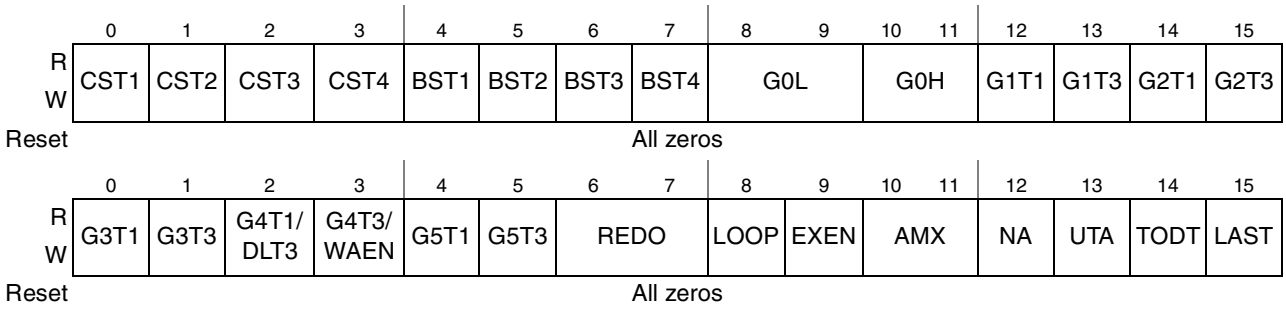


Figure 10-58. UPM RAM Word Field Descriptions

Table 10-30 describes RAM word fields.

Table 10-30. UPM RAM Word Field Descriptions

Bits	Name	Description
0	CST1	Chip select timing 1. Defines the state (0 or 1) of $\overline{\text{LCS}n}$ during bus clock quarter phase 1 if LCRR[CLKDIV] = 4 or 8. Defines the state (0 or 1) of $\overline{\text{LCS}n}$ during bus clock half phase 1 if LCRR[CLKDIV] = 2.
1	CST2	Chip select timing 2. Defines the state (0 or 1) of $\overline{\text{LCS}n}$ 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 $\overline{\text{LCS}n}$ during bus clock quarter phase 3 if LCRR[CLKDIV] = 4 or 8. Defines the state (0 or 1) of $\overline{\text{LCS}n}$ during bus clock half phase 2 if LCRR[CLKDIV] = 2.
3	CST4	Chip select timing 4. Defines the state (0 or 1) of $\overline{\text{LCS}n}$ during bus clock quarter phase 4 if LCRR[CLKDIV] = 4 or 8. Ignored when LCRR[CLKDIV] = 2.
4	BST1	Byte select timing1. Defines the state (0 or 1) of $\overline{\text{LBS}}$ during bus clock quarter phase 1 (LCRR[CLKDIV] = 4 or 8) or bus clock half phase 1 (LCRR[CLKDIV] = 2), in conjunction with BR*n*[PS] and LA[30:31].
5	BST2	Byte select timing 2:. Defines the state (0 or 1) of $\overline{\text{LBS}}$ during bus clock quarter phase 2 (LCRR[CLKDIV] = 4 or 8), in conjunction with BR*n*[PS] and LA[30:31]. Ignored when LCRR[CLKDIV] = 2.
6	BST3	Byte select timing 3. Defines the state (0 or 1) of $\overline{\text{LBS}}$ during bus clock quarter phase 3 (LCRR[CLKDIV] = 4 or 8) or bus clock half phase 2 (LCRR[CLKDIV] = 2), in conjunction with BR*n*[PS] and LA[30:31].
7	BST4	Byte select timing 4. Defines the state (0 or 1) of $\overline{\text{LBS}}$ during bus clock quarter phase 4 (LCRR[CLKDIV] = 4 or 8), in conjunction with BR*n*[PS] and LA[30:31]. Ignored when LCRR[CLKDIV] = 2.

Table 10-30. UPM RAM Word Field Descriptions (continued)

Bits	Name	Description
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 M*n*MR[G0CL] 01 Reserved 10 Asserted 11 Negated
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 M*n*MR[G0CL] 01 Reserved 10 Asserted 11 Negated
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 M*n*MR[GPL4]. If M*n*MR[GPL4] = 0 and LGPL4/LUPWAIT signal 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 M*n*MR[GPL4] = 1 and LGPL4/LUPWAIT functions as an input (LUPWAIT), if a read burst or single read is executed, G4T1/DLT3 defines the sampling of the data bus as follows: 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 M*n*MR[GPL4]. If M*n*MR[GPL4] = 0 and LGPL4/LUPWAIT signal 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 M*n*MR[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).

Table 10-30. UPM RAM Word Field Descriptions (continued)

Bits	Name	Description
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
24	LOOP	Loop start/end. The first RAM word in the RAM array where LOOP is 1 is recognized as the loop start word. The next RAM word where LOOP is 1 is the loop end word. RAM words between, and including the start and end words, are defined as part of the loop. The number of times the UPM executes this loop is defined in the corresponding loop fields of the M*n*MR. 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.
25	EXEN	Exception enable. Allows branching to an exception pattern at the exception start address (EXS). When an internal bus monitor time-out exception is recognized and EXEN in the RAM word is set, the UPM branches to the special exception start address (EXS) and begins operating as the pattern defined there specifies. 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 the 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. 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.
26–27	AMX	Address multiplexing. Determines the source of LAD[0:31] during a LALE phase. Any change in the AMX field initiates a new LALE (address) phase. 00 LAD[0:31] is the non-multiplexed address. For example, column address. 01 Reserved 10 LAD[0:31] is the address multiplexed according to M*n*MR[AM]. For example, row address. 11 LAD[0:31] is the contents of MAR. Used, for example, to initialize a mode. **Note:** Source ID debug mode is supported only for the AMX = 00 setting.
28	NA	Next burst address. Determines when the address is incremented during a burst access. 0 The address increment function is disabled. 1 The address is incremented in the next cycle. In conjunction with the BR*n*[PS], the increment value of LA[27:31] is 1, 2, or 4 for port sizes of 8-bits, 16-bits and 32-bits, respectively.
29	UTA	UPM transfer acknowledge. Indicates assertion of transfer acknowledge in the current cycle. 0 Transfer acknowledge is not asserted in the current cycle. 1 Transfer acknowledge is asserted in the current cycle.
30	TODT	Turn-on disable timer. The disable timer associated with each UPM allows a minimum time to be guaranteed between two successive accesses to the same memory bank. This feature is critical when DRAM requires a RAS precharge time. TODT turns the timer on to prevent another UPM access to the same bank until the timer expires.The disable timer period is determined in M*n*MR[DS*n*]. The disable timer does not affect memory accesses to different banks. Note that TODT must be set together with LAST, otherwise it is ignored. 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.

Table 10-30. UPM RAM Word Field Descriptions (continued)

Bits	Name	Description
31	LAST	Last word. When LAST is read in a RAM word, the current UPM pattern terminates and control signal timing set in the RAM word is applied to the current (and last) cycle. However, if the disable timer is activated and the next access is to the same bank, execution of the next UPM pattern is held off and the control signal values specified in the last word are extended in duration for the number of clock cycles specified in M*n*MR[DS*n*]. 0 The UPM continues executing RAM words. 1 Indicates the last RAM word in the program. Service to the UPM request is done after this cycle concludes.

10.4.4.4.2 Chip-Select Signal Timing (CST*n*)

If BR*n*[MSEL] of the accessed bank selects a UPM on the currently requested cycle, the UPM manipulates the $\overline{\text{LCS}n}$ for that bank with timing as specified in the UPM RAM word CST*n* fields. The selected UPM affects only the assertion and negation of the appropriate $\overline{\text{LCS}n}$ signal. The state of the selected $\overline{\text{LCS}n}$ signal of the corresponding bank depends on the value of each CST*n* bit. Figure 10-59 shows how UPMs control $\overline{\text{LCS}n}$ signals.

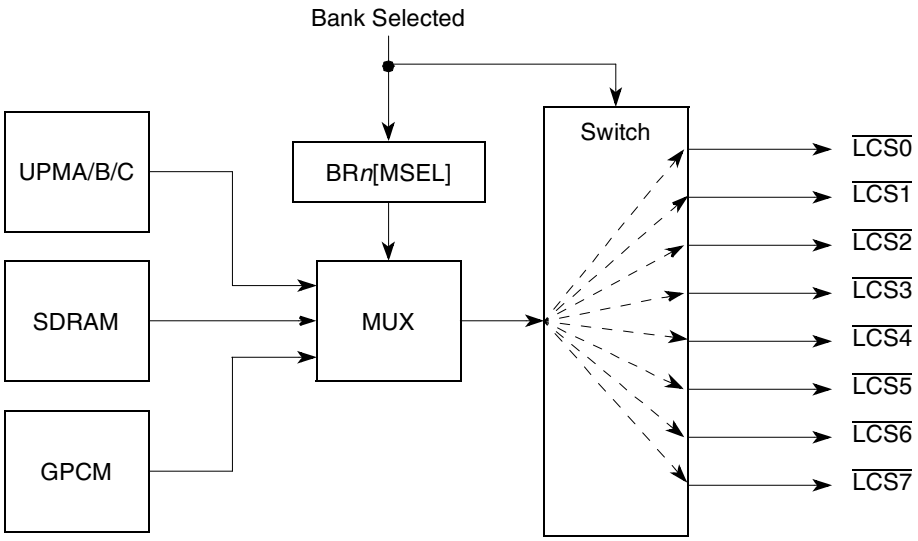


Figure 10-59. $\overline{\text{LCS}n}$ Signal Selection

10.4.4.4.3 Byte Select Signal Timing (BST*n*)

If BR*n*[MSEL] of the accessed memory bank selects a UPM on the currently requested cycle, the selected UPM affects the assertion and negation of the appropriate $\overline{\text{LBS}}$[0:3] signal. The timing of all four byte-select signals is specified in the RAM word. However, $\overline{\text{LBS}}$[0:3] are also controlled by the port size of the accessed bank, the number of bytes to transfer, and the address accessed.

Figure 10-60 shows how UPMs control $\overline{\text{LBS}}$[0:3].

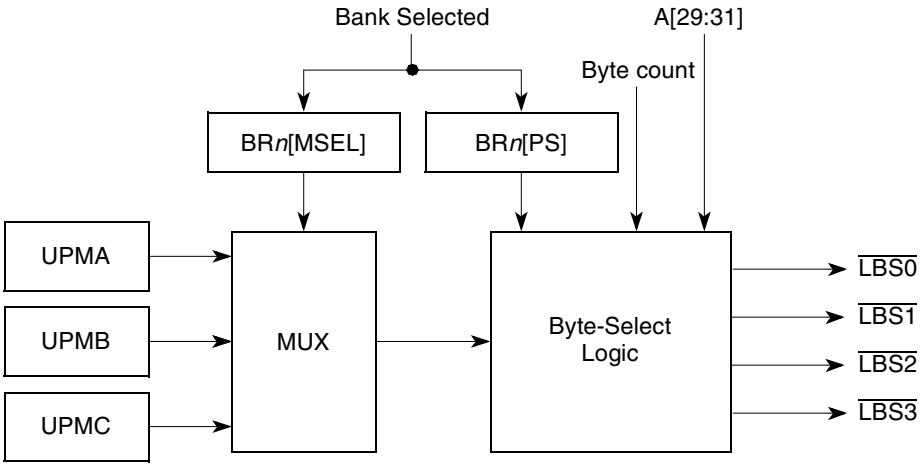


Figure 10-60. $\overline{\text{LBS}}$ Signal Selection

The uppermost byte select ($\overline{\text{LBS0}}$), when asserted, indicates that LAD[0:7] contains valid data during a cycle. Likewise, $\overline{\text{LBS1}}$ indicates that LAD[8:15] contains valid data, $\overline{\text{LBS2}}$ indicates that LAD[16:23] contains valid data, and $\overline{\text{LBS3}}$ indicates that LAD[24:31] contains valid data. For a UPM refresh timer request, all $\overline{\text{LBS}}$[0:3] signals are asserted/negated by the UPM according to the refresh pattern only. Following any internal bus monitor exception, the $\overline{\text{LBS}}$[0:3] signals are negated regardless of the exception handling provided by any UPM exception pattern to prevent spurious writes to external RAM.

10.4.4.4.4 General-Purpose Signals (G*n*T*n*, GO*n*)

The general-purpose signals (LGPL[0:5]) each have two bits in the RAM word that define the logical value of the signal to be changed at the rising edge of the bus clock and/or at the falling edge of the bus clock. LGPL0 offers enhancements beyond the other LGPL*n* lines.

GPL0 can be controlled by an address line specified in M*n*MR[G0CL]. To use this feature, G0H and G0L should be set in the RAM word. For example, for a SIMM with multiple banks, this address line can be used to switch between internal memory device banks.

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 10-31. The next RAM word for which LOOP = 1 is recognized as a loop end word. When it is reached, the loop counter is decremented by one.

Continued loop execution depends on the loop counter. If the counter is not zero, the next RAM word executed is the loop start word. Otherwise, the next RAM word executed is the one after the loop end word. Loops can be executed sequentially but cannot be nested. Also, special care must be taken: LAST and LOOP must not be set together.

Table 10-31. M*n*MR Loop Field Use

Request Serviced	Loop Field
Read single-beat cycle	RLF
Read burst cycle	RLF
Write single-beat cycle	WLF
Write burst cycle	WLF
Refresh timer expired	TLF
RUN command	RLF

10.4.4.4.6 Repeat Execution of Current RAM Word (REDO)

The REDO function is useful for wait-state insertion in a long 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 Address Multiplexing (AMX)

The address lines can be controlled by the pattern the user provides in the UPM. The address multiplex bits can choose between driving the transaction address, driving it according to the multiplexing specified by the M*n*MR[AM] field, or driving the MAR contents on the address signals. In all cases, LA[27:31] of the LBC are driven by the five lsbs of the address selected by AMX, regardless of whether the NA bit of the RAM word is used to increment the current address. The effect of NA = 1 is visible only when AMX = 00 chooses the column address.

Table 10-32 shows how M*n*MR[AM] settings affect address multiplexing when the RAM word AMX = 10. The 16 msbs of the LAD[0:31] bus during an address phase are driven with zero in the AMX = 10 case.

Table 10-32. UPM Address Multiplexing

AM	LAD[0:31] as Address Signals	A0–A15	A16	A17	A18	A19	A20	A21	A22	A23	A24	A25	A26	A27	A28	A29	A30	A31
000	Signal driven on external signal when address multiplexing is enabled— RAM word AMX = 10	0	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19	A20	A21	A22	A23
001		0	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19	A20	A21	A22
010		0	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19	A20	A21
011		0	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19	A20
100		0	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19
101		0	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18

Note that any change to AMX from one RAM word to the next RAM word executed results in an address phase on LAD[0:31] with the assertion of LALE for the number of cycles set for LALE in the OR*n* and LCRR registers. LGPL[0:5] signals maintain the value specified in the RAM word during the LALE phase.

10.4.4.4.8 Data Valid and Data Sample Control (UTA)

When a read access is handled by the UPM, and the UTA bit is 1 (data is to be sampled by the LBC), the value of the DLT3 bit in the same RAM word, in conjunction with M*n*MR[GPL*n*4DIS], determines when the data input is sampled by the LBC as follows:

- If M*n*MR[GPL*n*4DIS] = 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 LBC 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 GPL*n*4DIS = 0 (G4T4/DLT3 functions as G4T4), or if GPL*n*4DIS = 1 but DLT3 = 0, 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 10-61 shows how data sampling is controlled by the UPM.

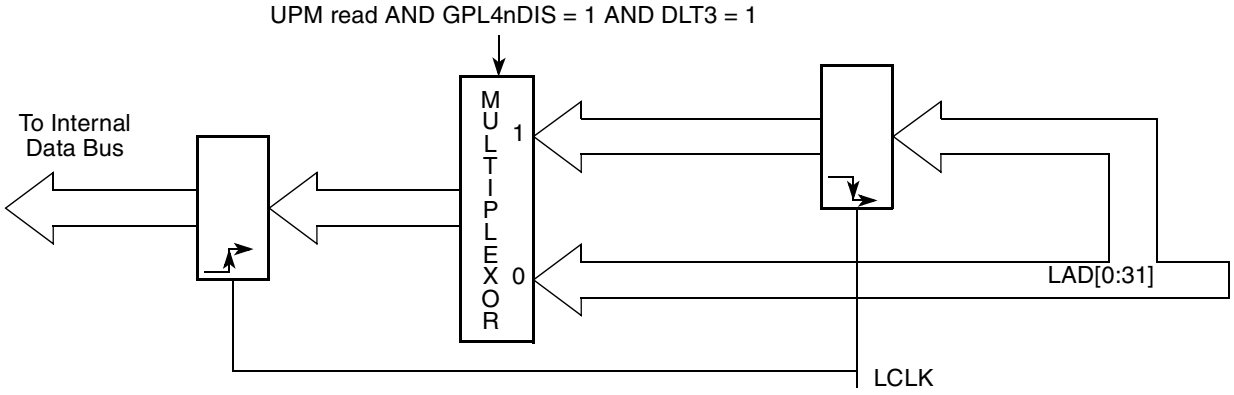


Figure 10-61. UPM Read Access Data Sampling

10.4.4.4.9 $\overline{\text{LGPL}}$[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 UPM signals are negated unconditionally (driven to logic one), 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.

10.4.4.4.10 Wait Mechanism (WAEN)

The WAEN bit in the RAM array word can be used to enable the UPM wait mechanism in selected UPM RAM words. If the UPM reads a RAM word with WAEN set, the external LUPWAIT signal is sampled and synchronized by the memory controller as if it were an asynchronous signal. The WAEN bit is ignored if LAST = 1 in the same RAM word.

Synchronization of LUPWAIT starts at the rising edge of the bus clock and takes at least 1 bus cycle to complete. If LUPWAIT is asserted and WAEN = 1 in the current UPM word, the UPM is frozen until LUPWAIT is negated. The value of external signals driven by the UPM remains as indicated in the previous RAM word. When LUPWAIT is negated, the UPM continues normal functions. Note that during WAIT cycles, the UPM does not handle data.

Figure 10-62 shows how the WAEN bit in the word read by the UPM and the LUPWAIT signal are used to hold the UPM in a particular state until LUPWAIT is negated. As the example shows, the $\overline{\text{LCS}n}$ and $\overline{\text{LGPL1}}$ 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.

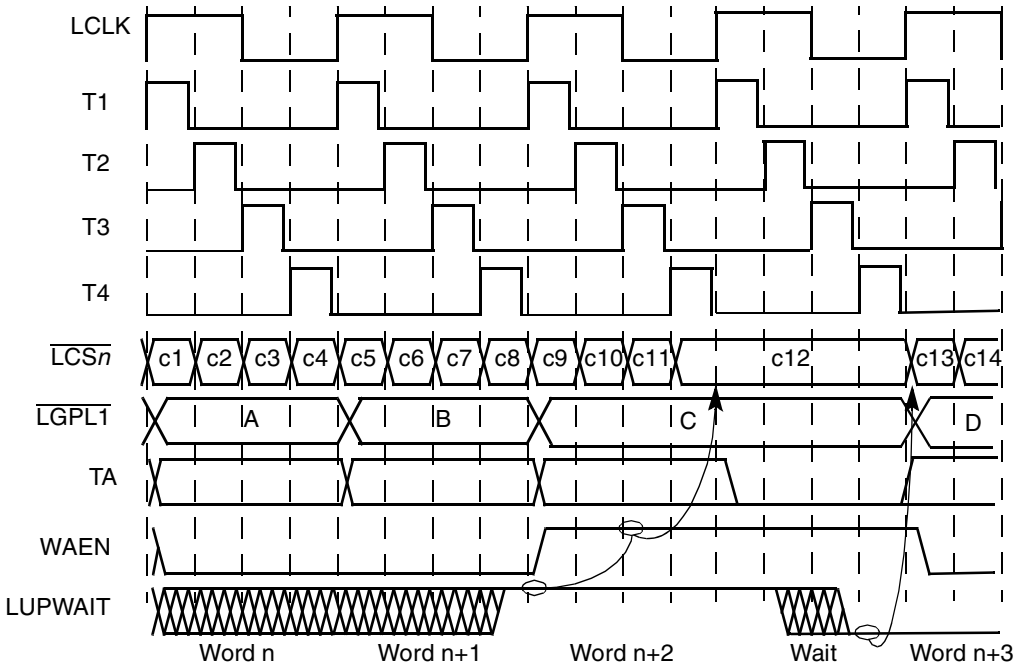


Figure 10-62. Effect of LUPWAIT Signal

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.

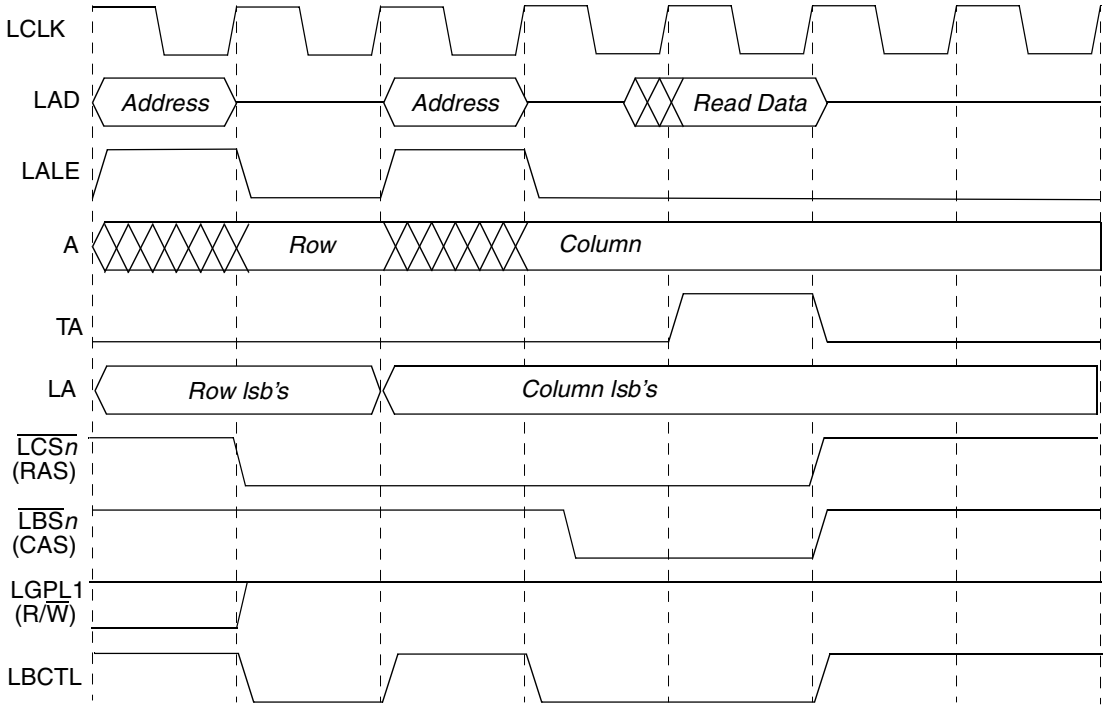
However, programming WAEN = 1 and UTA = 1 in the same RAM word allows the UPM to treat LUPWAIT as a synchronous signal, which must meet set-up and hold times in relation to the rising edge of the bus clock. In this case, as soon as UPM samples LUPWAIT negated on the rising edge of the bus clock, it immediately 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, and the acknowledge occurs regardless of whether the UPM was already frozen in WAIT cycles or not. This feature allows the synchronous negation of LUPWAIT to effect 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 access 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.

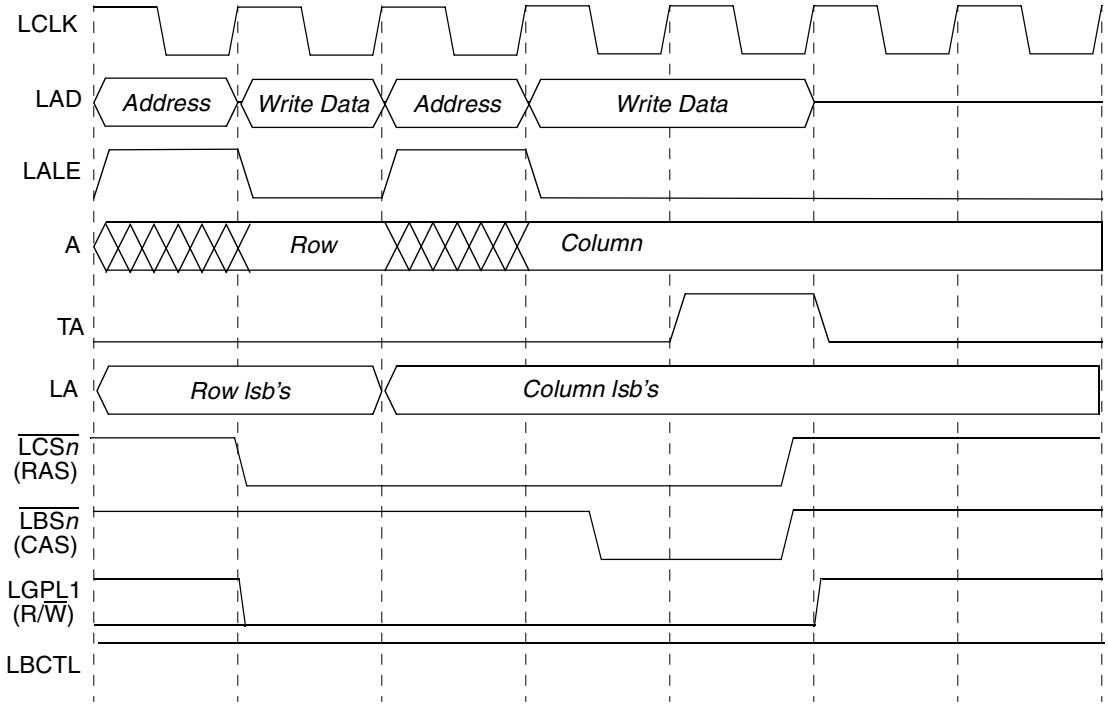
10.4.4.7 Memory System Interface Example Using UPM

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 presents timing diagrams for various UPM configurations, using fast-page mode DRAM as an example, with LCRR[CLKDIV] = 4 or 8. These examples may not represent the timing necessary for any specific device used with the LBC. Here, LGPL1 is programmed to drive R/$\overline{W}$ of the DRAM, although any LGPL*n* signal can be used for this purpose.



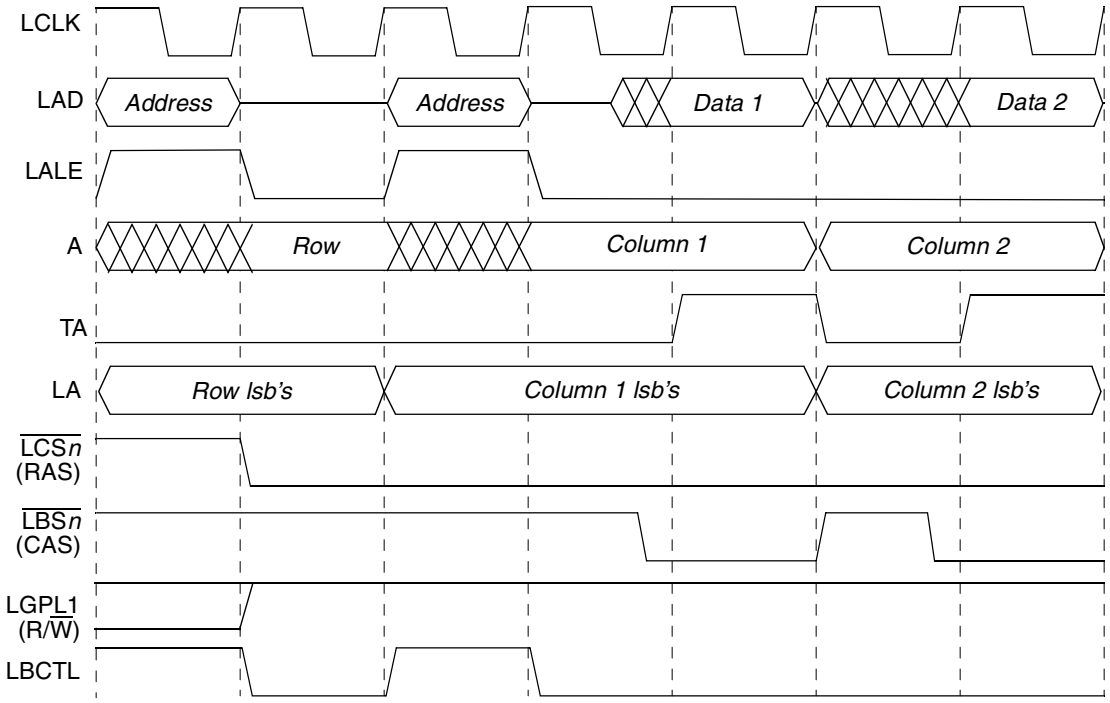
cst1 cst2 cst3 cst4	0 0 0 0	LALE pause (due to change in AMX)	0 0 0 0	0 0 0 0	Bit 0 Bit 1 Bit 2 Bit 3
bst1 bst2 bst3 bst4	1 1 1 1		1 0 0 0	0 0 0 0	Bit 4 Bit 5 Bit 6 Bit 7
g0l0 g0l1 g0h0 g0h1					Bit 8 Bit 9 Bit 10 Bit 11
g1t1 g1t3	1 1		1 1	1 1	Bit 12 Bit 13
g2t1 g2t3					Bit 14 Bit 15
g3t1 g3t3					Bit 16 Bit 17
g4t1 g4t3					Bit 18 Bit 19
g5t1 g5t3					Bit 20 Bit 21
redo[0] redo[1]					Bit 22 Bit 23
loop exen	0 0		0 0	0 0	Bit 24 Bit 25
amx0 amx1 na uta todt last	1 0 0 0 0 0		0 0 0 0 0 0	0 0 0 1 1 1	Bit 26 Bit 27 Bit 28 Bit 29 Bit 30 Bit 31
	RSS	**RSS+1**	**RSS+1**	**RSS+2**	

Figure 10-63. Single-Beat Read Access to FPM DRAM



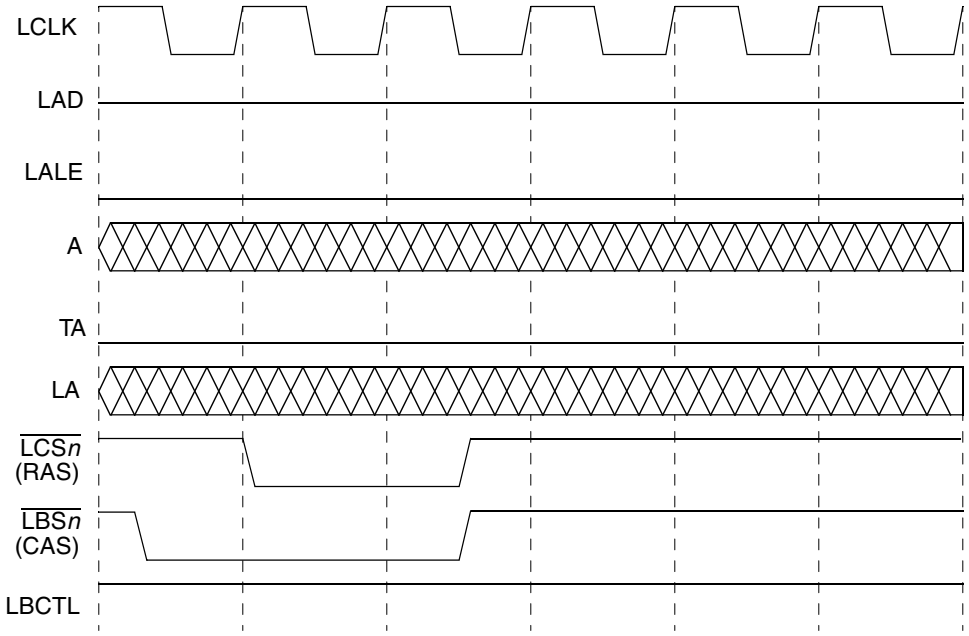
cst1 cst2 cst3 cst4	0 0 0 0	LALE pause (due to change in AMX)	0 0 0 0	0 0 0 1	Bit 0 Bit 1 Bit 2 Bit 3
bst1 bst2 bst3 bst4	1 1 1 1		1 1 0 0	0 0 0 1	Bit 4 Bit 5 Bit 6 Bit 7
g0l0 g0l1 g0h0 g0h1					Bit 8 Bit 9 Bit 10 Bit 11
g1t1 g1t3	0 0		0 0	0 0	Bit 12 Bit 13
g2t1 g2t3					Bit 14 Bit 15
g3t1 g3t3					Bit 16 Bit 17
g4t1 g4t3					Bit 18 Bit 19
g5t1 g5t3					Bit 20 Bit 21
redo[0] redo[1]					Bit 22 Bit 23
loop exen	0 0		0 0	0 0	Bit 24 Bit 25
amx0 amx1 na uta todt last	1 0 0 0 0 0		0 0 0 0 0 0	0 0 0 1 1 1	Bit 26 Bit 27 Bit 28 Bit 29 Bit 30 Bit 31
	WSS	**WSS+1**	**WSS+1**	**WSS+2**	

Figure 10-64. Single-Beat Write Access to FPM DRAM



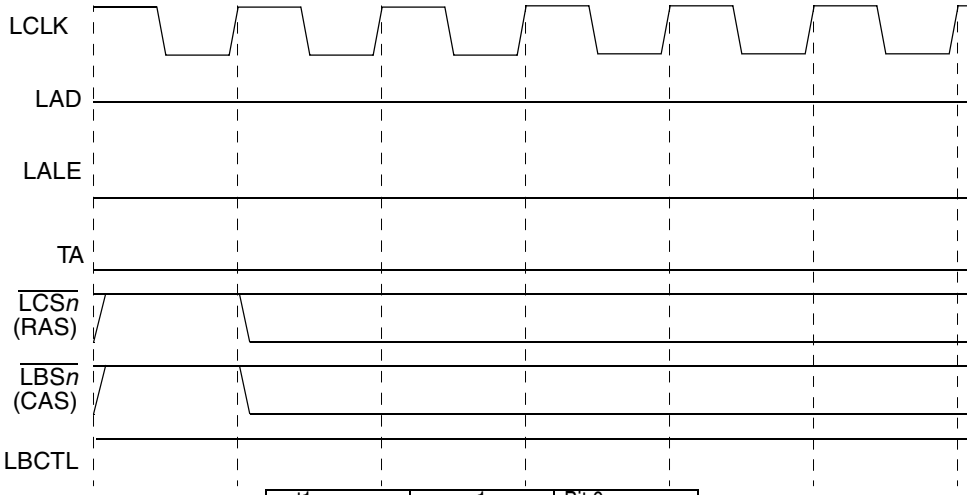
cst1	0	LALE pause (due to change in AMX)	0	0	1	Bit 0
cst2	0		0	0	1	Bit 1
cst3	0		0	0	1	Bit 2
cst4	0		0	0	1	Bit 3
bst1	1		1	0	1	Bit 4
bst2	1		1	0	1	Bit 5
bst3	1		1	0	1	Bit 6
bst4	1		0	0	1	Bit 7
g0l0						Bit 8
g0l1						Bit 9
g0h0						Bit 10
g0h1						Bit 11
g1t1	1		1	1	1	Bit 12
g1t3	1		1	1	1	Bit 13
g2t1						Bit 14
g2t3						Bit 15
g3t1						Bit 16
g3t3						Bit 17
g4t1						Bit 18
g4t3						Bit 19
g5t1						Bit 20
g5t3						Bit 21
redo[0]						Bit 22
redo[1]						Bit 23
loop	0		1	1	0	Bit 24
exen	0		0	1	0	Bit 25
amx0	1		0	0	0	Bit 26
amx1	0		0	0	0	Bit 27
na	0		0	1	0	Bit 28
uta	0		0	1	0	Bit 29
todt	0		0	0	1	Bit 30
last	0		0	0	1	Bit 31
	RBS		**RBS+1**	**RBS+2**	**RBS+3**	

Figure 10-65. Burst Read Access to FPM DRAM Using LOOP (Two Beats Shown)



cst1	1	0	0	Bit 0
cst2	1	0	0	Bit 1
cst3	1	0	1	Bit 2
cst4	1	0	1	Bit 3
bst1	1	0	0	Bit 4
bst2	0	0	0	Bit 5
bst3	0	0	1	Bit 6
bst4	0	0	1	Bit 7
g0l0				Bit 8
g0l1				Bit 9
g0h0				Bit 10
g0h1				Bit 11
g1t1				Bit 12
g1t3				Bit 13
g2t1				Bit 14
g2t3				Bit 15
g3t1				Bit 16
g3t3				Bit 17
g4t1				Bit 18
g4t3				Bit 19
g5t1				Bit 20
g5t3				Bit 21
redo[0]				Bit 22
redo[1]				Bit 23
loop	0	0	0	Bit 24
exen	0	0	0	Bit 25
amx0	0	0	0	Bit 26
amx1	0	0	0	Bit 27
na	0	0	0	Bit 28
uta	0	0	0	Bit 29
todt	0	0	1	Bit 30
last	0	0	1	Bit 31
	PTS	**PTS+1**	**PTS+2**	

Figure 10-66. Refresh Cycle (CBR) to FPM DRAM



cst1 cst2 cst3 cst4	1 1 1 1	Bit 0 Bit 1 Bit 2 Bit 3
bst1 bst2 bst3 bst4	1 1 1 1	Bit 4 Bit 5 Bit 6 Bit 7
g0l0 g0l1 g0h0 g0h1		Bit 8 Bit 9 Bit 10 Bit 11
g1t1 g1t3		Bit 12 Bit 13
g2t1 g2t3		Bit 14 Bit 15
g3t1 g3t3		Bit 16 Bit 17
g4t1 g4t3		Bit 18 Bit 19
g5t1 g5t3		Bit 20 Bit 21
redo[0] redo[1]		Bit 22 Bit 23
loop exen	0 0	Bit 24 Bit 25
amx0 amx1 na uta todt last	0 0 0 0 1 1	Bit 26 Bit 27 Bit 28 Bit 29 Bit 30 Bit 31
	EXS	

Figure 10-67. Exception Cycle

10.5 Initialization/Application Information

These sections provide detailed information on interfacing the LBC to peripheral devices.

10.5.1 Interfacing to Peripherals in Multiplexed Address/Data Mode

The local bus can be used either with separate address- and data buses or with a multiplexed address/data bus. These sections provide guidelines for interfacing peripherals in the multiplexed mode.

10.5.1.1 Multiplexed Address/Data Bus and Unmultiplexed Address Signals

To save signals on the local bus, address and data are multiplexed onto the same 32-bit bus. An external latch is needed to demultiplex and reconstruct the original address. No external intelligence is needed, because LALE provides the correct timing to control a standard logic latch. The LAD signals can be directly connected to the data signals of the memory/peripheral.

Transactions on the local bus start with an address phase, where the LBC drives the transaction address on the LAD signals and asserts the LALE signal. This can be used to latch the address and then the LBC can continue with the data phase.

The LBC supports port sizes of 8, 16, and 32 bits. For devices smaller than 32 bits, transactions must be broken down. For this reason, LA[30:31] are driven unmultiplexed. For 8-bit devices, LA[30:31] should be used and for 16 bit devices, LA30 should be used. 32-bit devices use neither of these signals.

In addition, the LBC supports burst transfers (not in the GPCM machine). LA[27:29] are the burst addresses within a natural 32-byte burst. To minimize the amount of address phases needed on the local bus and to optimize the throughput, those signals are driven separately and should be used whenever a device requires the 5 least-significant addresses. Those should not be used from LAD[27:31].

All other addresses, A[0:26], must be reconstructed through the latch.

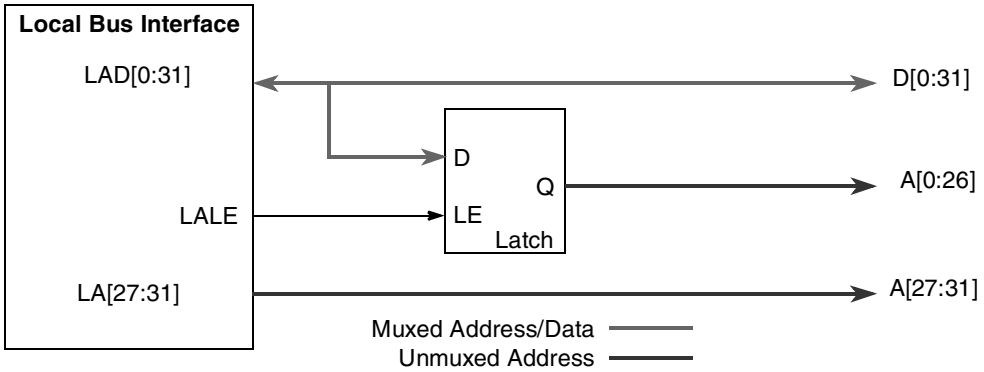


Figure 10-68. Multiplexed Address/Data Bus

10.5.1.2 Peripheral Hierarchy on the Local Bus

To achieve high bus speed interfaces for synchronous SRAMs or SDRAMs, a hierarchy of the memories/peripherals connected to the local bus is suggested, as shown in Figure 10-69.

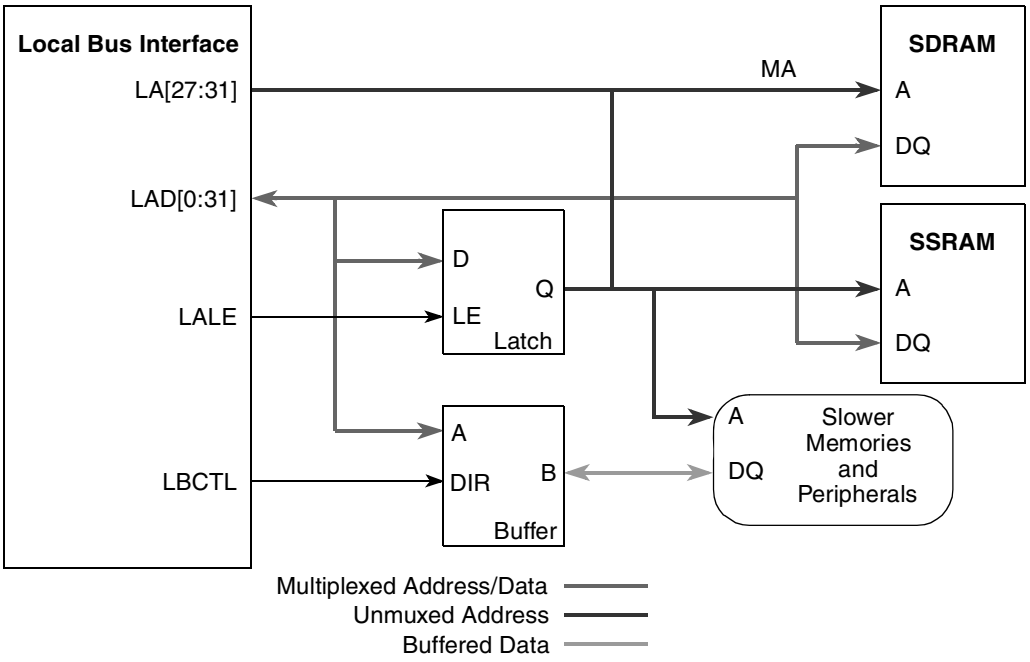


Figure 10-69. Local Bus Peripheral Hierarchy

The multiplexed address/data bus sees the capacitive loading of the data signals of the fast SDRAMs or synchronous SRAMs plus one load for an address latch plus one load for a buffer to the slow memories. The loadings of all other memories and peripherals are hidden behind the buffer and the latch. The system designer needs to investigate the loading scenario and ensure that I/O timings can be met with the loading determined by the connected components.

10.5.1.3 Peripheral Hierarchy on the Local Bus for Very High Bus Speeds

To achieve the highest possible bus speeds on the local bus, it is recommended to reduce the number of devices connected directly to the local bus even further. For those cases probably only one bank of synchronous SRAMs or SDRAMs should be used and instead of using a separate latch and a separate bus transceiver, a bus demultiplexer combining those two functions into one device should be used. Figure 10-70 shows an example of such a hierarchy. This section is only a guideline; the board designer must simulate the electric characteristics of the chosen scenario to determine the maximum operating frequency.

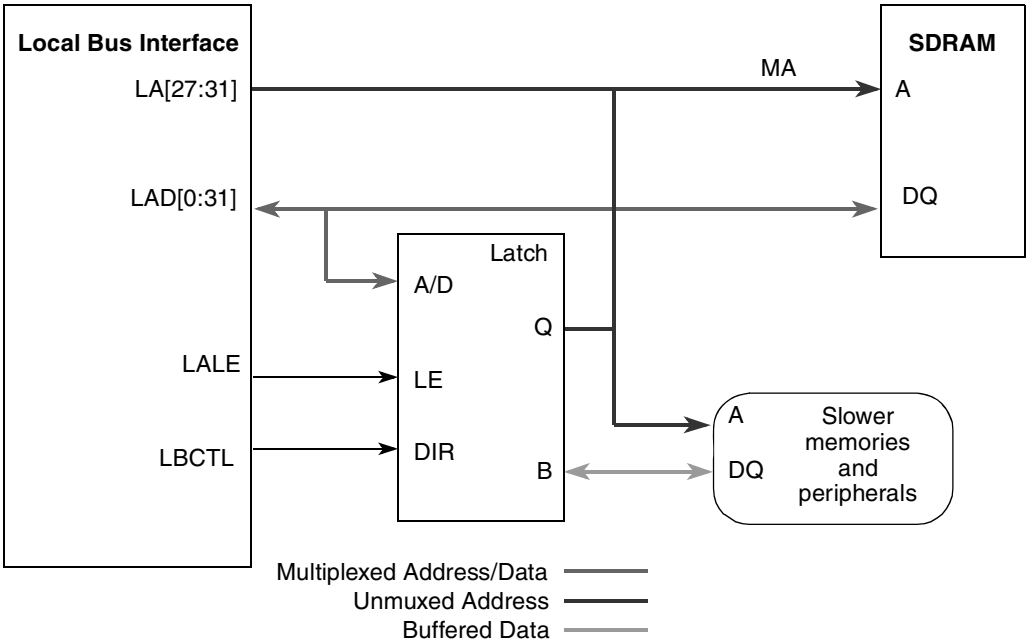


Figure 10-70. Local Bus Peripheral Hierarchy for Very High Bus Speeds

10.5.1.4 GPCM Timings

If a system contains a memory hierarchy with high-speed synchronous memories (SDRAM, synchronous SRAM) and lower speed asynchronous memories (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. The GPCM address timings is shown in Figure 10-71.

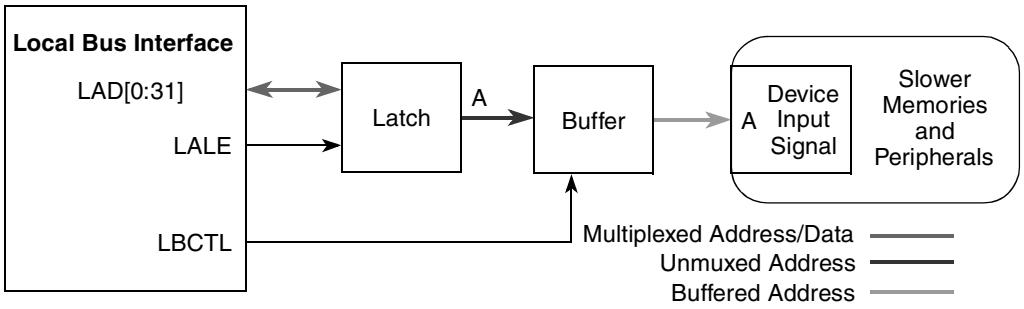


Figure 10-71. GPCM Address Timings

To calculate address setup timing for a slower peripheral/memory device, several parameters have to be added: propagation delay for the address latch, propagation delay for the buffer and the address setup for the actual peripheral. Typical values for the two propagation delays are in the order of 3–6 ns, so for a 166-MHz bus frequency, $\overline{\text{LCS}}$ should arrive roughly 3 bus clocks later.

For data timings, only the propagation delay of one buffer plus the actual data setup time must be considered.

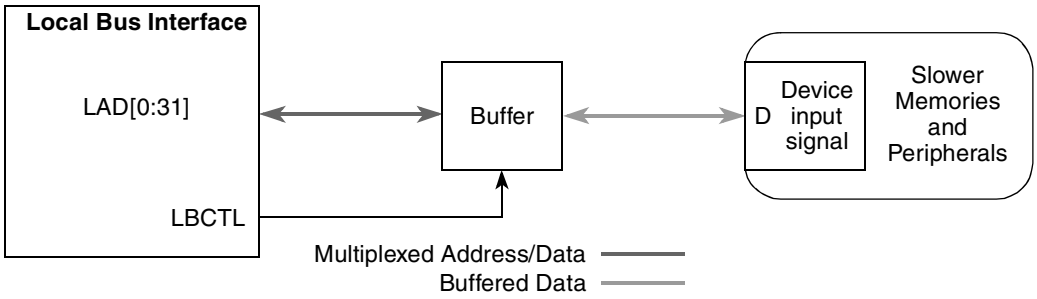


Figure 10-72. GPCM Data Timings

10.5.2 Bus Turnaround

Because the local bus uses multiplexed address and data, special consideration must be given to avoid bus contention at bus turnaround. The following cases must be examined:

- Address phase after previous read
- Read data phase after address phase
- Read-modify-write cycle for parity protected memory banks
- UPM cycles with additional address phases

The bus does not change direction for the following cases, so they need no special attention:

- Continued burst after the first beat
- Write data phase after address phase
- Address phase after previous write

10.5.2.1 Address Phase after Previous Read

During a read cycle, the memory/peripheral drives the bus and the bus transceiver drives LAD. After the data has been sampled, the output drivers of the external device must be disabled. This can take some time; for slow devices, the EHTR feature of the GPCM or the programmability of the UPM should be used to guarantee that those devices have stopped driving the bus when the LBC memory controller ends the bus cycle.

In this case, after the previous cycle ends, LBCTL goes high and changes the direction of the bus transceiver. The LBC then inserts a bus turnaround cycle to avoid contention. The external device has now already placed its data signals in high impedance and no bus contention will occur.

10.5.2.2 Read Data Phase after Address Phase

During the address phase, LAD actively drives the address and LBCTL is high, driving the bus transceivers in the same direction as during a write. After the end of the address phase, LBCTL goes low and changes the direction of the bus transceiver. The LBC places the LAD signals in high impedance after its t_{dis}(LB). The LBCTL will have its new state after t_{en}(LB) and, because this is an asynchronous input, the transceiver starts to drive those signals after its t_{en}(transceiver) time. The system designer has to ensure, that [t_{en}(LB) + t_{en}(transceiver)] is larger than t_{dis} (LB) to avoid bus contention.

10.5.2.3 Read-Modify-Write Cycle for Parity Protected Memory Banks

Principally, a read-modify-write cycle is a read cycle immediately followed by a write cycle. Because the write cycle has a new address phase in any case, this basically is the same case as an address phase after a previous read.

10.5.2.4 UPM Cycles with Additional Address Phases

The flexibility of the UPM allows the user to insert additional address phases during read cycles by changing the AMX field, therefore turning around the bus during one pattern. The LBC automatically inserts a single bus turnaround cycle if the bus (LAD) was previously high impedance for any reason, such as a read, before LALE is driven and LAD is driven with the new address. The turnaround cycle is not inserted on a write, because the bus was already driven to begin with.

However, bus contention could potentially still occur on the far side of a bus transceiver. It is the responsibility of the designer of the UPM pattern to guarantee that enough idle cycles are inserted in the UPM pattern to avoid this.

10.5.3 Interface to Different Port-Size Devices

The LBC supports 8-, 16-, and 32-bit data port sizes. However, the bus requires that the portion of the data bus used for a transfer to or from a particular port size be fixed. A 32-bit port must reside on D[0:31], a 16-bit port must reside on D[0:15], and an 8-bit port must reside on D[0:7]. The local bus always tries to transfer the maximum amount of data on all bus cycles. Figure 10-73 shows the device connections on the data bus.

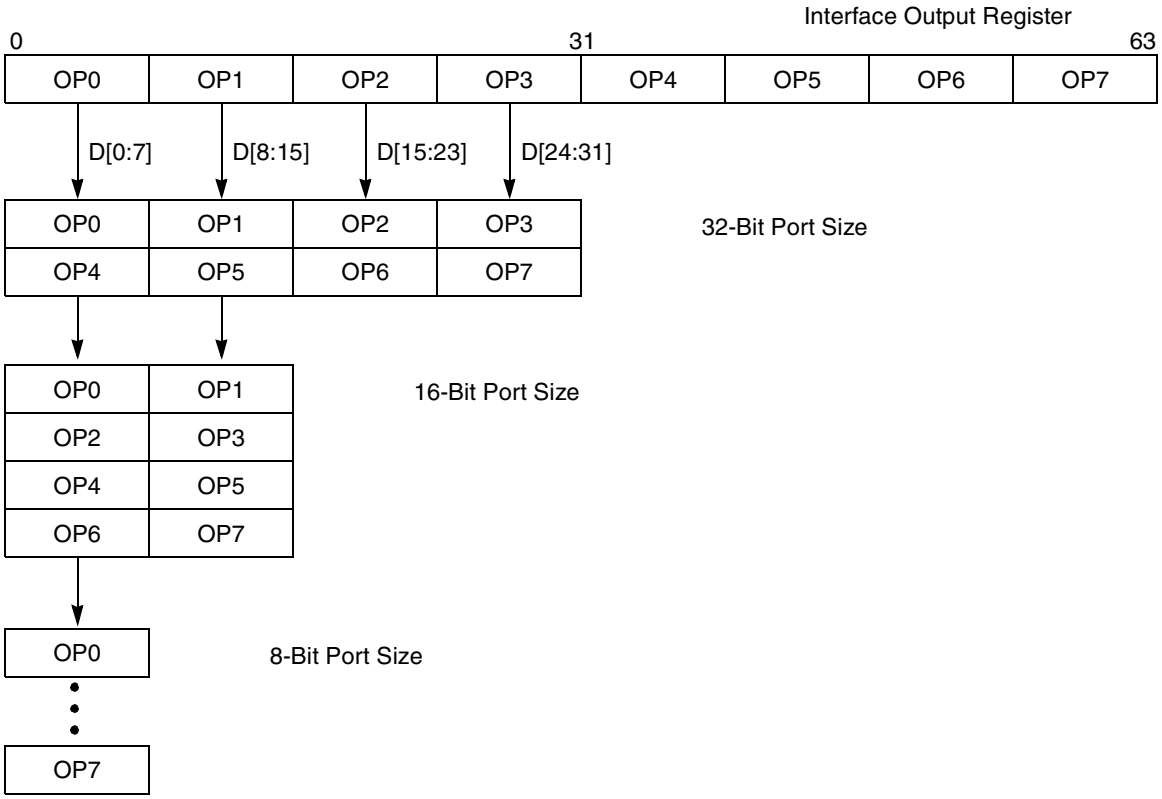


Figure 10-73. Interface to Different Port-Size Devices

Table 10-33 lists the bytes required on the data bus for read cycles.

Table 10-33. Data Bus Requirements For Read Cycle

Transfer Size	Address State [1] A[29:31]	Port Size/Data Bus Assignments						
		32-Bit				16-Bit		8-Bit
		0–7	8–15	16–23	24–31	0–7	8–15	0–7
Byte	000	OP0[2]	—[3]	—	—	OP0	—	OP0
	001	—	OP1	—	—	—	OP1	OP1
	010	—	—	OP2	—	OP2	—	OP2
	011	—	—	—	OP3	—	OP3	OP3
	100	OP4	—	—	—	OP4	—	OP4
	101	—	OP5	—	—	—	OP5	OP5
	110	—	—	OP6	—	OP6	—	OP6
	111	—	—	—	OP7	—	OP7	OP7

Table 10-33. Data Bus Requirements For Read Cycle (continued)

Transfer Size	Address State [1] A[29:31]	Port Size/Data Bus Assignments						
		32-Bit				16-Bit		8-Bit
		0–7	8–15	16–23	24–31	0–7	8–15	0–7
Half Word	000	OP0	OP1	—	—	OP0	OP1	OP0
	001	—	OP1	OP2	—	—	OP1	OP1
	010	—	—	OP2	OP3	OP2	OP3	OP2
	100	OP4	OP5	—	—	OP4	OP5	OP4
	101	—	OP5	OP6	—	—	OP5	OP5
	110	—	—	OP6	OP7	OP6	OP7	OP6
Word	000	OP0	OP1	OP2	OP3	OP0	OP1	OP0
	100	OP4	OP5	OP6	OP7	OP4	OP5	OP4

1 Address state is the calculated address for port size.

2 OP*n*: These lanes are read or written during that bus transaction. OP0 is the most-significant byte of a word operand and OP3 is the least-significant byte.

3 — Denotes a byte not driven during that write cycle.

10.5.4 Interfacing to SDRAM

The following subsections provide application information on interfacing to SDRAM.

10.5.4.1 Basic SDRAM Capabilities of the Local Bus

The LBC provides one SDRAM machine for the local bus. Although there is only one machine, multiple chip selects ($\overline{\text{LCS}}n$) can be programmed to support multiple SDRAM devices. Note that no limitation exists on the number of chip selects that can be programmed for SDRAM. This means that $\overline{\text{LCS}}$[1:7] can be programmed to support SDRAM, assuming $\overline{\text{LCS0}}$ is reserved for the GPCM to connect to Flash memory.

If multiple chip selects are configured to support SDRAM on the local bus, each SDRAM device should have the same port size and timing parameters. This means that all option registers (OR*n*) for the SDRAM chip selects should be programmed the same.

NOTE

Although in principle it is possible to mix different port sizes and timing parameters, combinations are limited and this operation is not recommended.

All the chip selects share the same local bus SDRAM mode register (LSDMR) for initialization along with the local bus-assigned SDRAM refresh timer register (LSRT) and the memory refresh timer prescaler register (MPTPR) for refresh.

For refresh, the memory controller supplies auto refresh to SDRAM according to the time interval specified in LSRT and MPTPR as follows:

$$\text{Refresh Period} = \frac{\text{LSRTx (MPTPR[PTP])}}{\text{System Frequency}}$$

This represents the time period required between refreshes. When the refresh timer expires, the memory controller issues a CBR to each chip select. Each CBR is separated by one clock. A refresh timing diagram for multiple chip selects is shown in Figure 10-51 in Section 10.4.3.11.1, "SDRAM Refresh Timing."

During a memory transaction dispatched to the local bus, the memory controller compares the memory address with the address information of each chip select (programmed with BR*n* and OR*n*). If the comparison matches a chip select that is controlled by SDRAM, the memory controller requests service to the local bus SDRAM machine, depending on the information in BR*n*. Although multiple chip selects may be programmed for SDRAM, only one chip select is active at any given time; thus, multiple chip selects can share the same SDRAM machine.

10.5.4.2 Maximum Amount of SDRAM Supported

Table 10-34 summarizes information based on SDRAM data sheets supplied by Micron.

Table 10-34. Micron SDRAM Devices

SDRAM Device	64 Mbit				128 Mbit				256 Mbit				512 Mbit			
I/O Port	x4	x8	x16	x32	x4	x8	x16	x32	x4	x8	x16	x32	x4	x8	x16	x32
Bank	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
Row	12	12	12	11	12	12	12	12	13	13	13	13	13	13	13	TBD
Column	10	9	8	8	11	10	9	8	11	10	9	8	12	11	10	TBD

The data port size is programmable, but the following examples use all 32 bits of the local bus. The 32-bit port size requires four SDRAM devices (with 8-bit I/O ports) connected in parallel to a single chip select. If 128-Mbit devices are used, one chip select provides 128-Mbit/device×4 devices = 64 Mbytes. If 4 chip selects are programmed for SDRAM use, the result is 64 Mbytes×4 = 256 Mbyte. If 256-Mbit SDRAM devices are used, the total available memory is 512 Mbyte. Consequently 512-Mbit devices allow for 1 Gbyte.

Although there is no technical difficulty in supporting multiple chip select configurations, in practice, the user may want to maximize the amount of SDRAM assigned to each chip select to minimize cost.

10.5.4.3 SDRAM Machine Limitations

This section describes limitations of the local bus SDRAM machine.

10.5.4.3.1 Analysis of Maximum Row Number Due to Bank Select Multiplexing

LSDMR[BSMA] is used to multiplex the bank select address. The BSMA field and corresponding multiplexed address are shown below:

000 LA12–LA13
001 LA13–LA14
…
111 LA19–LA20

Note that LA12 is the latched value of LAD12.

The highest address signals that the bank selects can be multiplexed with are LA[12:13], which limits the signals for the row address to LA[14:31]. For a 32-bit port, the maximum width of the local bus, LA[30:31] are not connected, and the maximum row is LA[14:29]. The local bus SDRAM machine supports 15 rows, which is sufficient for all devices.

10.5.4.3.2 Bank Select Signals

Page-based interleaving allows bank signals to be multiplexed to the higher-order address signals to leave room for future upgrades. For example, a user could multiplex the bank select signals to LA[14:15], leaving LA16 to connect to the address signal for a larger memory size.

This allows the system designer to design one board that can be used with a current generation of SDRAM devices and upgraded to the next generation without requiring a new board layout.

10.5.4.3.3 128-Mbyte SDRAM

Figure 10-74 shows the connection to an SDRAM of 128 Mbytes. Note that all circuit diagrams are principal connection diagrams and do not show any means of signal integrity.

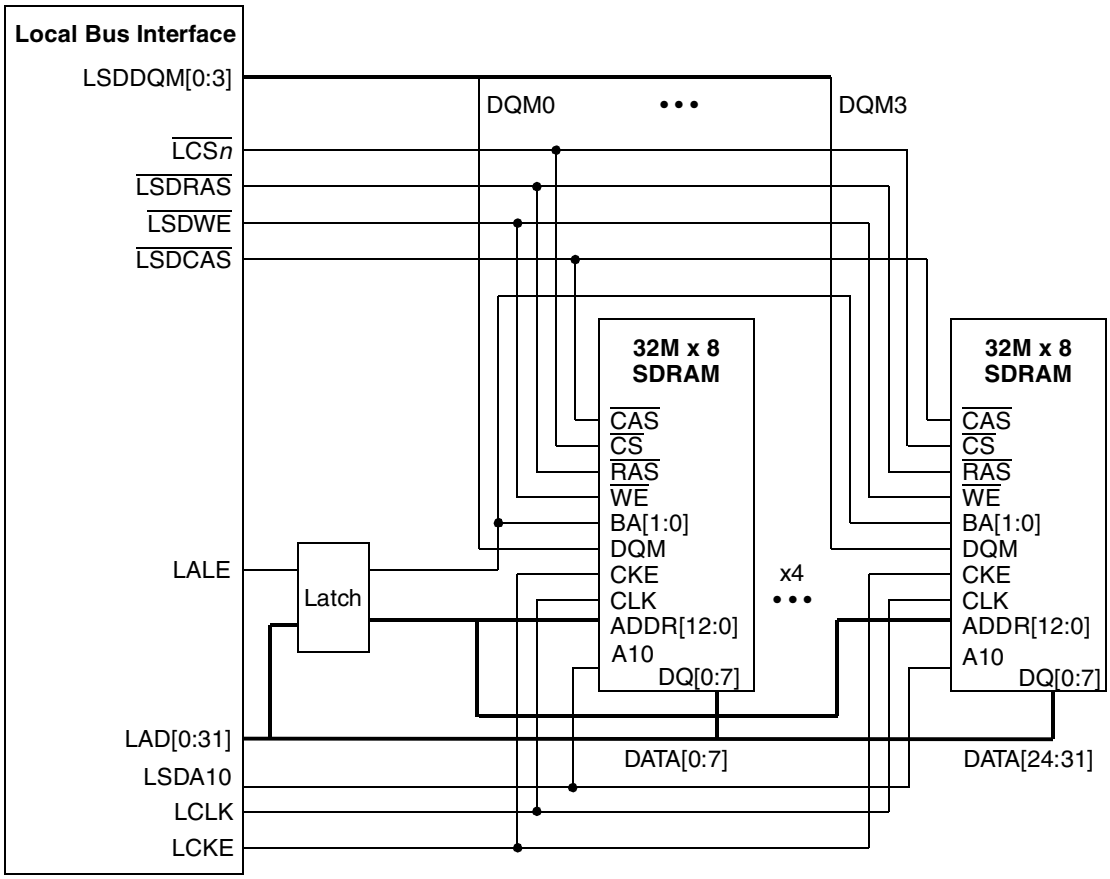


Figure 10-74. 128-Mbyte SDRAM Diagram

Table 10-35 shows details about LAD*n* signal connections for the example in Figure 10-74.

Table 10-35. LAD*n* Signal Connections to 128-Mbyte SDRAM

LAD (Latch Address)	SDRAM Address Signal
LAD29	A0
LAD28	A1
LAD27	A2
LAD26	A3
LAD25	A4
LAD24	A5
LAD23	A6
LAD22	A7

Table 10-35. LAD*n* Signal Connections to 128-Mbyte SDRAM (continued)

LAD (Latch Address)	SDRAM Address Signal
LAD21	A8
LAD20	A9
LAD19 (no connect)	A10 is connected to LSDA10
LAD18	A11
LAD17	A12
LAD16	BA0 (if LSDMR[BSMA] = 011)
LAD15	BA1 (if LSDMR[BSMA] = 011)

Consider the following SDRAM organization:

- The 32-bit port size is organized as 8 x 8 x 32 Mbit.
- Each device has four internal banks, 13 row address lines, and 10 column address lines.

The logical address is partitioned as shown in Table 10-36.

Table 10-36. Logical Address Bus Partitioning

A[0:4]	A[5:17]	A[18:19]	A[20:29]	A[30:31]
msb of start address	Row	Bank select	Column	lsb

The following parameters are extracted:

- COLS = 011, 10 column lines
- ROWS = 100, 13 row lines

During the address phase, the SDRAM address port is set as shown in Table 10-37.

Table 10-37. SDRAM Device Address Port During Address Phase

LA[0:14]	LA[15:16]	LA[17:29]	LA[30:31]
—	Internal bank select A[18:19]	Row A[5:17]	no-connect

Because the internal bank selects are multiplexed over LA[15:16], LSDMR[BSMA] must be set to 011.

Table 10-38 shows the address port configuration during a READ/WRITE command.

Table 10-38. SDRAM Device Address Port During READ/WRITE Command

LA[0:14]	LA[15:16]	LA[17:18]	LA[19]	LA[20:29]	LA[30:31]
msb of start address	Internal bank select	Don't care	AP	Column	no-connect

Table 10-39 shows the register configuration for this example. PSRT and MPTPR are not shown but should be programmed according to the device's specific refresh requirements.

Table 10-39. Register Settings for 128-Mbytes SDRAMs

Register	Field	Value
BR*n*	BA	Base address
	PS	11 = 32-bit port size
	MS	011 = SDRAM-local bus
	V	1
OR*n*	AM	11_1111_1000_0000_0000_0
	COLS	011
	ROWS	100
LSDMR	RFEN	1
	OP	000
	BSMA	011
	RFRC	From device data sheet
	PRETOACT	From device data sheet
	ACTTOROW	From device data sheet
	BL	0
	WRC	From device data sheet
	BUFCMD	0
	CL	From device data sheet

10.5.4.3.4 256-Mbyte SDRAM

This example uses the same SDRAM as the previous example, but doubles the number of devices connected and therefore uses two chip selects.

10.5.4.3.5 512-Mbyte SDRAM

This example uses the MT48LC64M4A2FB from Micron to implement 512 Mbytes.

In this SDRAM organization:

- The 32-bit port size is 8 x 4 x 64 Mbit x 2 chip select lines.
- Each device has 4 internal banks, 13 row address lines, and 11 column address lines.

The logical address is partitioned as shown in Table 10-40.

Table 10-40. Logical Address Partitioning

A[0:3]	A[4:16]	A[17:18]	A[19:29]	A[30:31]
msb of start address	Row	Bank select	Column	lsb

The following parameters can be extracted:

- COLS = 100, 11 column lines
- ROWS = 100, 13 row lines

During the address phase, the SDRAM address port is set as in Table 10-41.

Table 10-41. SDRAM Device Address Port During Address Phase

LA[0:13]	LA[15:16]	LA[17:29]	LA[30:31]
—	Internal bank select (A[17:18])	Row (A[4:16])	no-connect

Because the internal bank selects are multiplexed over LA[15:16], LSDMR[BSMA] must be set to 011.

Table 10-42 shows the address port settings during a READ/WRITE command.

Table 10-42. SDRAM Device Address Port During READ/WRITE Command

LA[0:14]	LA[15:16]	LA[17]	LA[18]	LA[19:29]	LA[30:31]
msb of start address	Internal bank select	Don't care	AP	Column	no-connect

Table 10-43 shows the register configuration. PSRT and MPTPR are not shown, but they should be programmed according to the specific device's refresh requirements.

Table 10-43. Register Settings for 512-Mbyte SDRAMs

Register	Field	Value
BR*n*	BA	Base address
	PS	11 = 32-bit port size
	MS	011 = SDRAM-local bus
	V	1
OR*n*	AM	11_1110_0000_0000_0000_0
	BPD	01
	COLS	100
	ROWS	100
PSDMR	RFEN	1
	OP	000
	BSMA	011
	RFRC	From device data sheet
	PRETOACT	From device data sheet
	ACTTOROW	From device data sheet
	BL	0
	WRC	From device data sheet
	BUFCMD	0
	CL	From device data sheet

10.5.4.4 Parity Support for SDRAM

Unlike older DRAM technologies, SDRAM devices typically are organized either x4, x8, x16 or x32. No mainstream devices include parity support. To allow for error protection on the local bus an additional SDRAM for the 4 parity bits must be used. Because the local bus allows for SDRAM accesses with less than the full port size, read-modify-write cycles are supported for SDRAM write cycles.

Figure 10-75 shows a connection diagram.

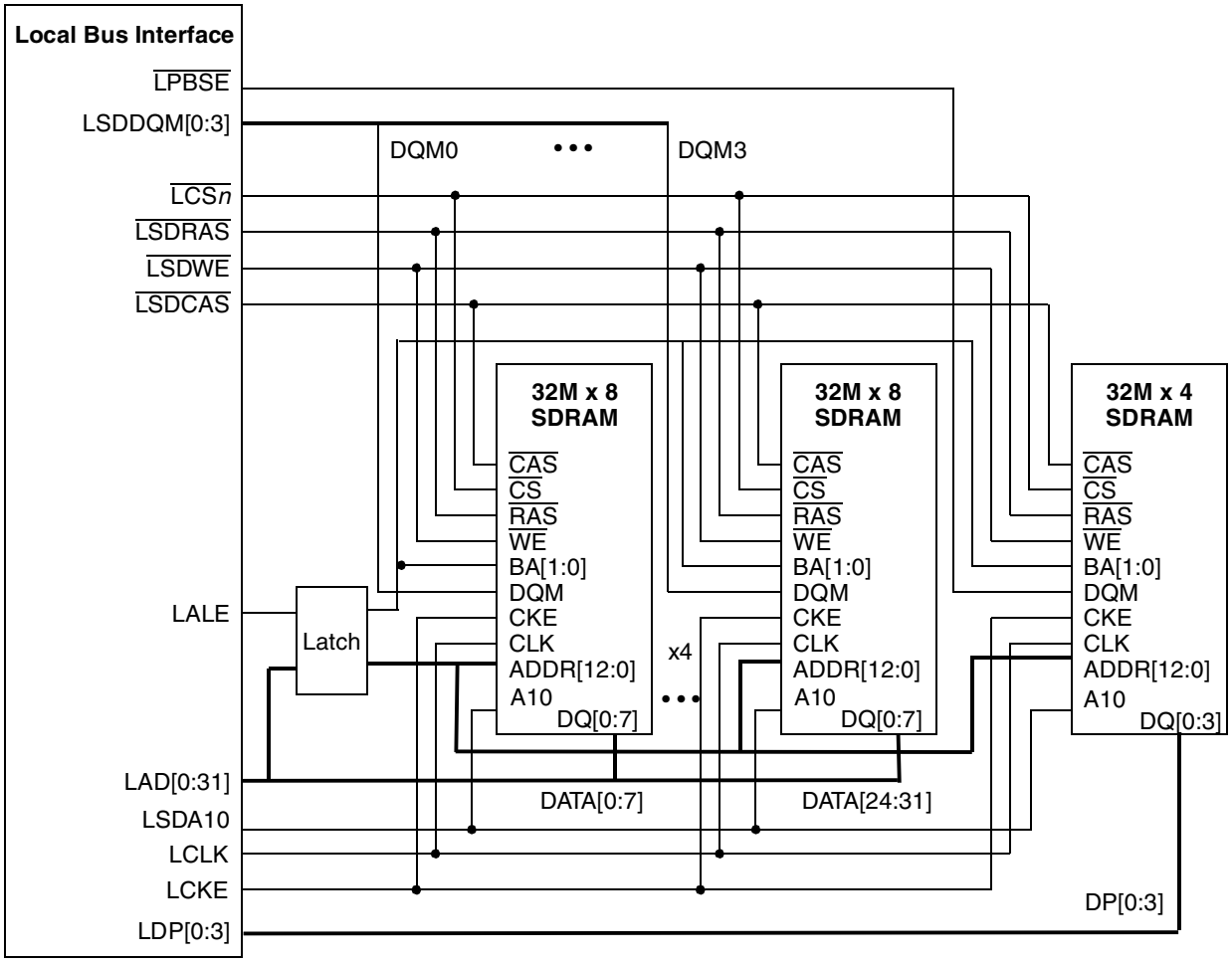


Figure 10-75. Parity Support for SDRAM

10.5.5 Interfacing to ZBT SRAM

In many applications, SDRAM provides sufficient performance for the local bus. However, especially in networking applications, memory access patterns are often random and SDRAM is not optimized for that case. ZBT SRAMs have been designed to optimize the performance in networking applications. This section describes how to interface to ZBT SRAMs. Figure 10-76 shows the connections. The UPM is used to generate control signals. The same interfacing is used for pipelined and flow-through versions of ZBT SRAMs. However different UPM patterns must be generated for those cases. Because ZBT SRAMs are used mostly by performance-critical applications, a 32-bit maximum local bus width is assumed.

ZBT SRAMs allow different configurations. For the local bus, the burst order should be set to linear burst order by tying the mode signal to GND; $\overline{\text{CKE}}$ should also be tied to ground.

ZBT SRAMs perform four-beat bursts. Because the LBC generates eight-beat transactions (for 32-bit ports) the UPM breaks down each burst into two consecutive four-beat bursts. The internal address generator of the LBC generates the new A27 for the second burst.

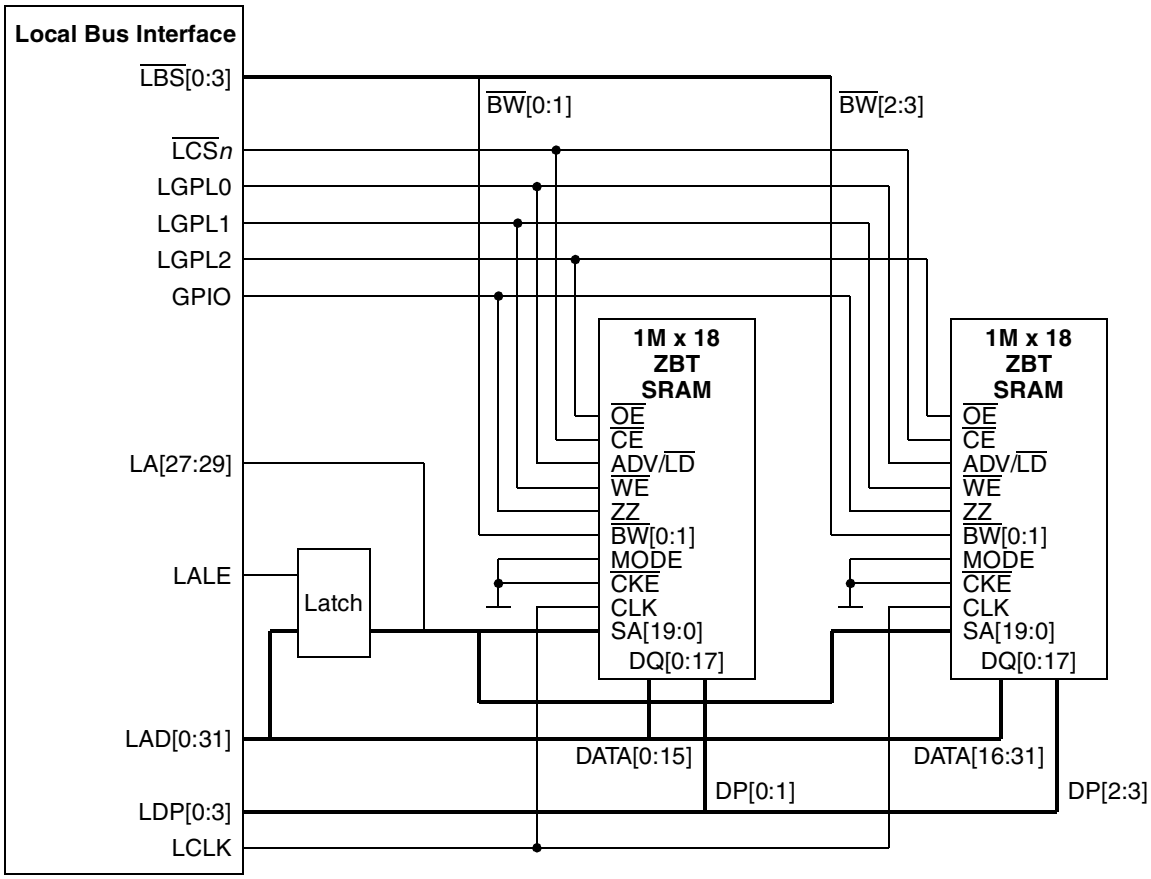


Figure 10-76. Interface to ZBT SRAM

Because linear burst on the SRAM is used, the device itself bursts with the burst addresses of [0:1:2:3]. The local bus always generates linear bursts and expects [0:1:2:3:4:5:6:7]. Therefore, two consecutive linear bursts of the ZBT SRAM with A27 = 0 for the first burst and A27 = 1 for the second burst give the desired burst pattern.

The UPM also supports single beat accesses. Because the ZBT SRAM does not support this and always responds with a burst, the UPM pattern must 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 must wait for the end of the SRAM burst to avoid bus contention with further bus activities.

Note that SRAMs are available with natural parity. In the example, a ×18 SRAM, which holds two data bytes and two parity bits, is used. While for the support of parity on SDRAM banks, the local bus must use read-modify-write cycles and compromise performance, SRAM banks can be used with natural parity and do not compromise performance for parity support.

10.5.6 Interfacing to DSP Host Ports

In many applications, an integrated communications processor aggregates traffic for DSPs and distributes that traffic to the DSPs. The local bus allows connection to a variety of different DSP host ports and this section gives some information on how to interface to some example DSPs.

10.5.6.1 Interfacing to MSC8122 DSI

The MSC8122 direct-slave interface (DSI) gives an external host direct access to the MSC8122. It provides the following slave interfaces to an external host:

- Asynchronous SRAM-like interface giving the host single accesses (with no external clock).
- Synchronous SSRAM-like interface giving the host single or burst accesses of 256 bits (eight beats of 32 bits or four beats of 64 bits) with its external clock decoupled from the MSC8122 internal bus clock.

The DSI supports 32- or 64-bit data bus modes. For connection to the local bus, the DSI must be configured in 32-bit mode. As part of the DSP reset.

The DSI supports two addressing modes, which are determined during the MSC8122 boot sequence and are used in both 32- and 64-bit data modes. Refer to details in the MSC8122 documentation.

- Full address bus mode with HA[11–29]
- Sliding window mode with HA[14–29]

10.5.6.2 DSI in Asynchronous SRAM-Like Mode

The local bus supports the DSI single- and double-strobe of operation. The dual strobe configuration in Figure 10-77 shows the interface to the MSC8122 DSI for asynchronous mode.

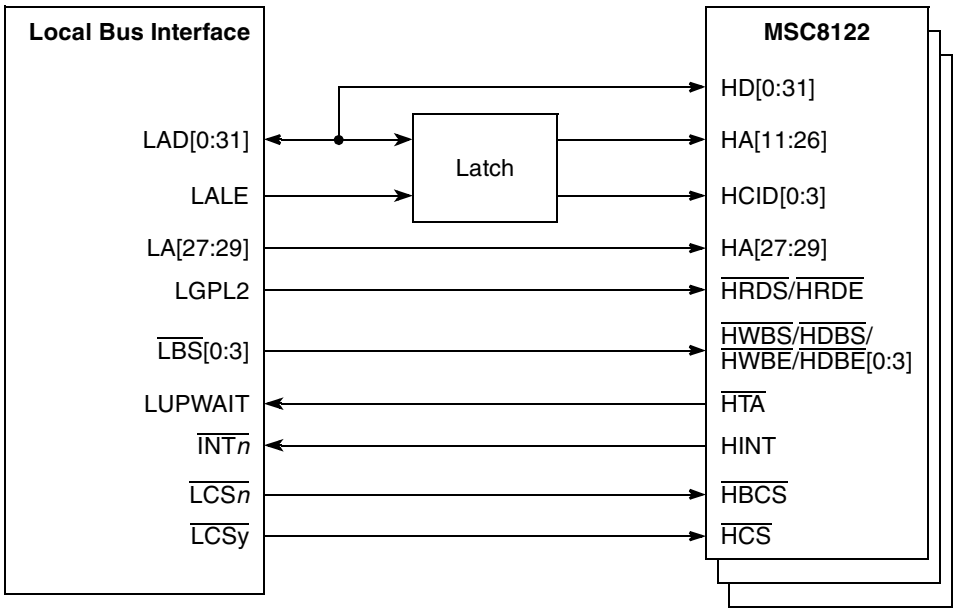


Figure 10-77. Interface to MSC8122 DSI in Asynchronous Mode

The asynchronous SRAM-like mode of the DSI is inherently slower than the synchronous mode and should be used if only relatively small amounts of data are transferred between the communications controller and the MSC8122. For maximum timing flexibility, the UPM machine of the LBC should be used.

The UPM programmer is responsible for ensuring correct setup and hold timings for all signals. The UPM allows sufficient control to satisfy any requirements here.

Figure 10-78 shows an asynchronous write access. The DSI samples the host chip ID signals (HCID[0:3]) on the first falling edge of the host write byte strobe signals ($\overline{\text{HWBS}}$) on which the host chip select signal ($\overline{\text{HCS}}$) is asserted. If HCID[0:3] match the CHIPID value, the DSI is accessed. By asserting $\overline{\text{HTA}}$, the DSI signals whether it is ready to sample the host data bus (HD), and the host can terminate the access by immediately negating $\overline{\text{HWBS}}$. The WAEN feature of the UPM must be used to insert wait states while the DSI is busy. M*n*MR[UWPL] must be cleared to interpret the correct polarity of $\overline{\text{HTA}}$. The DSI samples the host address bus (HA) and the host data bus (HD) on the rising edge of $\overline{\text{HWBS}}$. In addition, the assertion of $\overline{\text{HWBS}}$[0:3] are sampled at the end and are part of the access attributes.

Because the UPM is used for this mode, the DCR[HTAAD] should be set 1 and DCR[HTADT] should be defined to a non zero value. This mode is to be used in implementations with a pull-up resistor on $\overline{\text{HTA}}$. The host can start its next access (back-to-back accesses) without negating $\overline{\text{HCS}}$ between accesses. If the next access is not to the same MSC8122, to prevent contention on $\overline{\text{HTA}}$, the host must wait until the previous DSI stops driving $\overline{\text{HTA}}$ before it accesses the next device. If the next access is to the same MSC8122, the host must not start consecutive accesses before $\overline{\text{HTA}}$ is asserted by the previous access. The

easiest way to achieve this is to insert idle cycles at the end of the UPM pattern to guarantee that $\overline{\text{HTA}}$ is inactive.

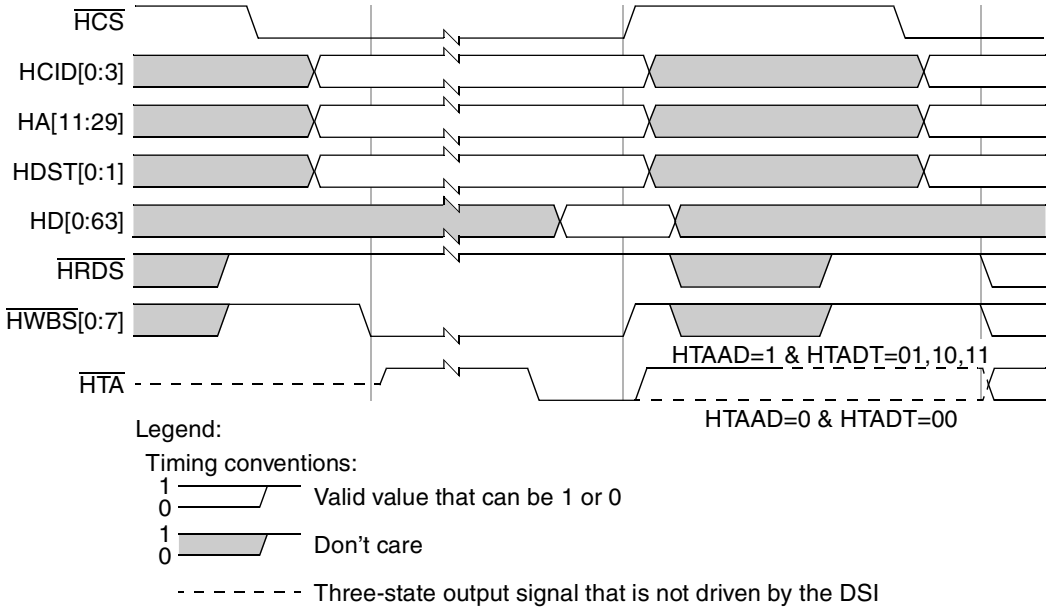


Figure 10-78. Asynchronous Write to MSC8122 DSI

Figure 10-79 shows an asynchronous read access. The DSI samples the host address bus (HA) and the HCID on the first falling edge of the host read strobe signal ($\overline{\text{HRDS}}$) on which the $\overline{\text{HCS}}$ is asserted. If HCID[0:3] match the CHIPID value, the DSI is accessed. When the DCR[RPE] bit is set, read access to the memory space (not to the register space) initiates data prefetching from consecutive addresses in the internal memory space. By asserting $\overline{\text{HTA}}$, the DSI signals that the data is valid and that the host can sample the host data bus (HD) and terminate the access by negating $\overline{\text{HRDS}}$. If the data for this access is already in the read buffer due to the prefetch mechanism, assertion time of $\overline{\text{HTA}}$ is improved. The WAEN feature of the UPM must be used to insert wait states while the DSI is busy. M*n*MR[UWPL] has to be cleared to interpret the correct polarity of the $\overline{\text{HTA}}$ signal.

Because the UPM is used in the mode, the DCR[HTAAD] should be set 1 and the drive time control field, DCR[HTADT], should be defined to a non zero value. This mode is specially designed to be used for implementations with a pull-up resistor on $\overline{\text{HTA}}$.

The host can start its next access (back-to-back accesses) without negating $\overline{\text{HCS}}$ between accesses. If the next access is not to the same MSC8122, then to prevent contention on $\overline{\text{HTA}}$, the host must wait until the previous DSI stops driving $\overline{\text{HTA}}$ before it accesses the next device. If the next access is to the same MSC8122, the host must not start consecutive access before $\overline{\text{HTA}}$ is actively driven to 1 by the previous access. The easiest way to achieve this is to insert idle cycles at the end of the UPM pattern to guarantee that $\overline{\text{HTA}}$ is inactive.

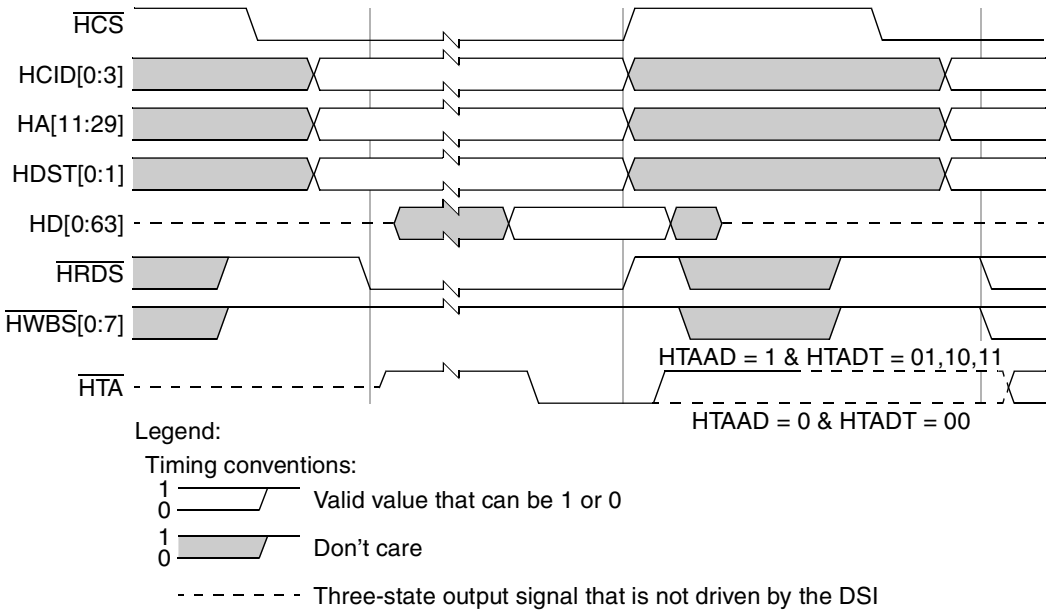


Figure 10-79. Asynchronous Read from MSC8122 DSI

10.5.6.3 DSI in Synchronous Mode

The synchronous SSRAM-like mode of the DSI is inherently faster than the asynchronous mode and should be used if larger amounts of data are transferred between the communications controller and the MSC8122. This will optimize the bus utilization, especially if several MSC8122s are connected to one local bus. The UPM machine of the LBC must be used to implement this interface.

Figure 10-80 shows the interface for synchronous mode. Because the DSI asserts and negates $\overline{\text{HTA}}$ in synchronous mode even within a burst transfer on a clock-by-clock basis, and because the DSI expects the host to react within one clock cycle, some tricks can be implemented to support the synchronous mode.

$\overline{\text{HTA}}$ drives LUPWAIT of the UPM. M*n*MR[UWPL] must be cleared to interpret the correct polarity of $\overline{\text{HTA}}$. Because this signal influences the internal state machine of the local bus clock, the local bus cannot react to $\overline{\text{HTA}}$ changes correctly within one local bus clock. Refer to Section 10.4.4.4.10, "Wait Mechanism (WAEN)."

The solution to this lies in that the local bus operates at a higher frequency than the DSI interface of the DSP. The local bus clock can be divided by an integer divider (1:2, 1:3 or 1:4) to generate the DSI clock. This should not be a problem because the local bus is designed for much higher frequencies than the DSI. Because all timings are given in DSP DSI clock cycles, the UPM patterns must be adjusted appropriately and need to assert a signal for 2, 3, or 4 clocks (as many as the divider ratio) instead of one. Fortunately, the UPM has the REDO feature, which allows every UPM RAM entry to be executed 1x, 2x, 3x, or 4x, which should be sufficient for any likely divider ratio.

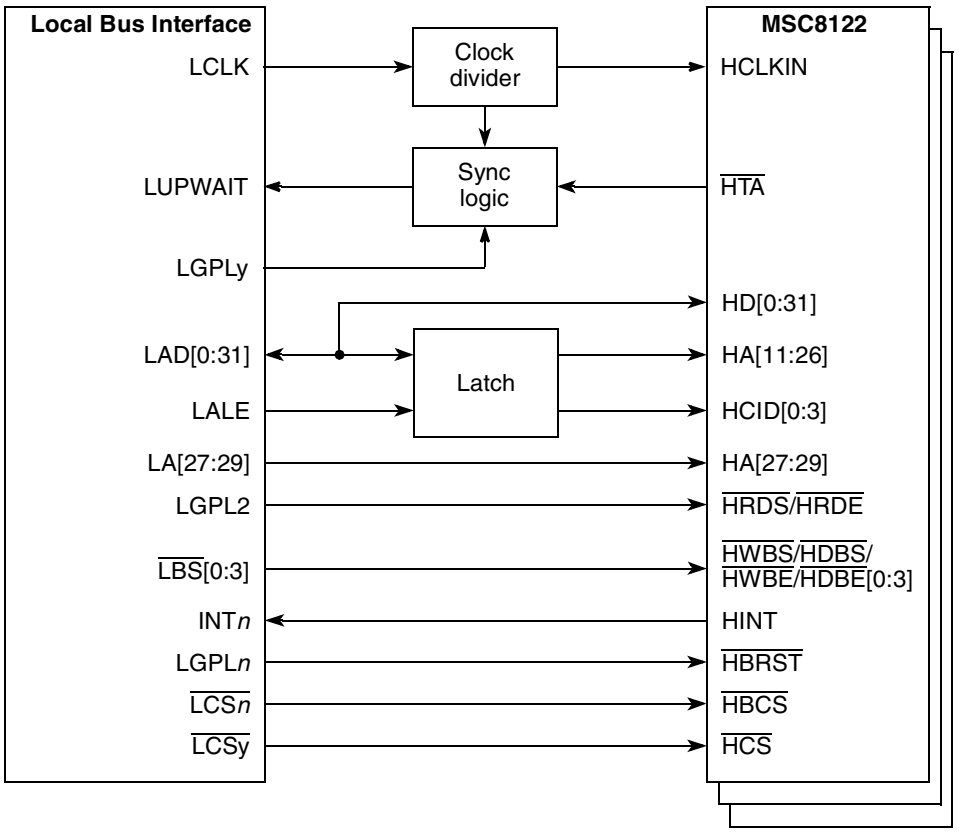


Figure 10-80. Interface to MSC8122 DSI in Synchronous Mode

This solution allows the local bus to react within multiple local bus clocks to the HTA signal and be still within one DSI clock. Typically, LUPWAIT is synchronized internal, and only 2 clocks after LUPWAIT changes, new data can be sampled or presented. For example, if the local bus clock ratio is 3x the DSI clock, data can be sampled in the third local bus sub-clock, which is the last third of the DSI clock. If the local bus clock ratio is only 2x the DSI clock, there is a special mode, where the LUPWAIT is not synchronized. Refer to Section 10.4.4.5, "Synchronous Sampling of LUPWAIT for Early Transfer Acknowledge." In this mode, data is sampled in the 2nd subclock, which is the second half of the DSI clock. AC timing of LUPWAIT must be met in this mode; otherwise behavior is indeterminate.

The remaining issue is the synchronization of the UPM cycles to the beginning of the DSI clock cycle. Because the UPM executes *n* cycles for every DSI cycle, UPM transitions must be synchronized to the DSI clock. The solution is to use a special synchronize cycle at the beginning of the pattern. A GPL signal controls a multiplexer and activates external synchronization logic, which uses the DSI clock to stall the UPM by asserting LUPWAIT until the beginning of the next DSI cycle. After that, this GPL signal must be negated and the multiplexer connects LUPWAIT to HTA instead, for the rest of the bus cycle. Note that the GPL signals should be used in the inverted state of their inactive state (GPL[0:4] are 1 when inactive, GPL5 is 0 when inactive) to start the synchronization process.

Figure 10-81 shows such a synchronization mechanism for a clock divider of 3. Note that the length of the synchronization cycle depends on the relative start of the synchronization process and varies with every access. It can vary from one to *n* (clock ratio) local bus clocks.

The second column (compensation cycle) is intended to compensate for the reaction time of LUPWAIT to get in lock step with the DSI clock. For example, if the clock divider ratio is 1:3 and the LUPWAIT reaction time is two local bus clocks, because LUPWAIT is synchronized, then one local bus clock should be inserted.

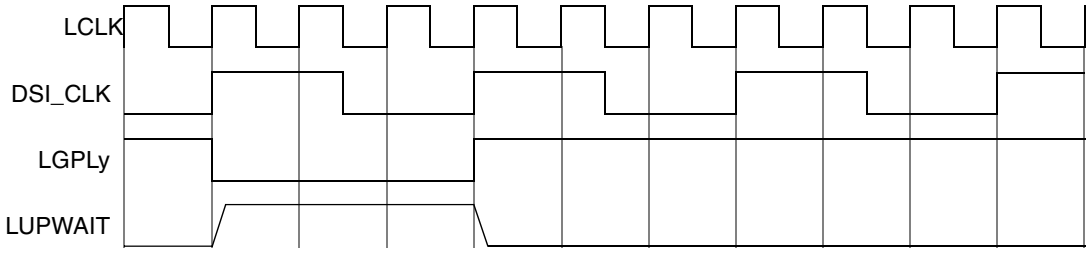


Figure 10-81. UPM Synchronization Cycle

Table 10-44. UPM Synchronization Cycles

	Synchronization Cycle	Compensation Cycle	DSI Cycle 1	Bits
cst1–cst4				0–3
bst1–bst4				4–7
g0xx				8–11
g1tx				12–13
g2tx				14–15
g3tx				16–17
g4t1				18
g4t3	1	0		19
g5tx				20–21
redo[0]	0	0	1	22
redo[1]	0	0	0	23
loop	0	0		24
exen	0			25
amx0	0	0		26
amx1	0	0		27
na	0			28
uta	0			29
todt	0			30
last	0			31

This section describes synchronous single write and read, and synchronous burst write and read operations.

The local bus supports the DSI single strobe as well as the DSI double strobes of operation. The dual strobe configuration is shown as an example.

10.5.6.3.1 Synchronous Single Write

Figure 10-82 shows a synchronous single write access.

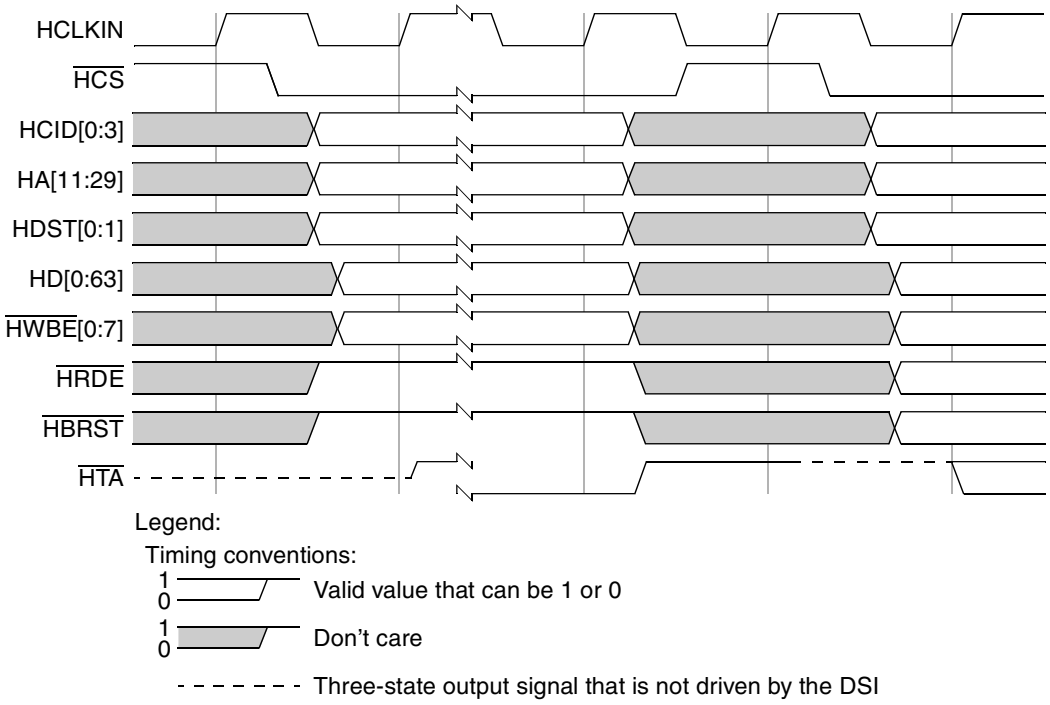


Figure 10-82. Synchronous Single Write to MSC8122 DSI

The DSI samples HA, HDST, HCID, HD, HWBE, HRDE, and HBRST on the first HCLKIN rising edge on which HCS is asserted. If HCID[0:3] match the CHIPID value, the DSI is accessed. At least one HWBE signal is asserted, and HRDE and HBRST are negated. Assertion of HTA indicates that the DSI is ready to complete the current access and the host must terminate this access. Because HTA is connected to the LUPWAIT signal of the UPM, all local bus signals are frozen until HTA goes to 0 and then the UPM continues in its pattern. Typically, HTA is asserted immediately. If the write buffer is full, HTA assertion is delayed. HTA is asserted for one HCLKIN cycle, driven to logic 1 in the next cycle, and stops being driven on the next rising edge of HCLKIN. The host can start its next access to the same MSC8122 immediately on the next HCLKIN rising edge without negating HCS between accesses. If the next access is not to the same MSC8122, then, to prevent contention on HTA, the host must wait to access the next device until the previous DSI stops driving HTA. The easiest way to achieve this is to insert idle cycles at the end of the UPM pattern to guarantee that HTA is inactive.

10.5.6.3.2 Synchronous Single Read

Figure 10-83 shows a synchronous single read access.

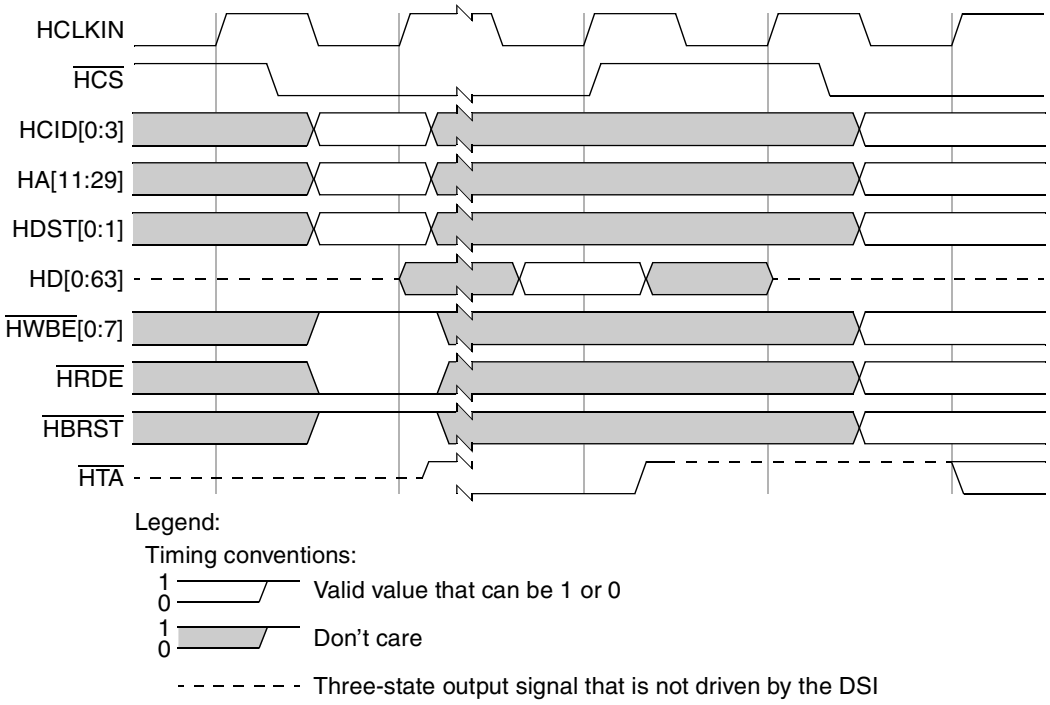


Figure 10-83. Synchronous Single Read from MSC8122 DSI

The DSI samples HA, HDST, HCID, HWBE, HRDE, and HBRST on the first HCLKIN rising edge on which HCS is asserted. If the HCID[0:3] signals match the CHIPID value, the DSI is accessed. HRDE is asserted, and HWBE and HBRST are negated. If DCR[RPE] is set (see MSC8122 documentation), read access to the memory space (not to the register space) initiates prefetching data from consecutive addresses in the internal memory space. Assertion of HTA indicates that data is valid and the host must sample the HD and terminate the access. Because HTA is connected to the UPM LUPWAIT signal, all local bus signals are frozen until HTA goes to 0; then the UPM continues in its pattern. HTA is asserted earlier when the data for this access is already prefetched to the read buffer. It asserted for one HCLKIN cycle and driven to logic 1 in the next cycle. It stops being driven on the next rising edge of HCLKIN. The host can start its next access to the same MSC8122 immediately in the next HCLKIN rising edge without negating HCS between accesses. If the next access is not to the same MSC8122, then, to prevent contention on HTA, the host must wait to access the next device until the previous DSI stops driving HTA. The easiest way to achieve this is to insert idle cycles at the end of the UPM pattern to guarantee that HTA is inactive.

10.5.6.3.3 Synchronous Burst Write

Figure 10-84 shows a synchronous burst write access.

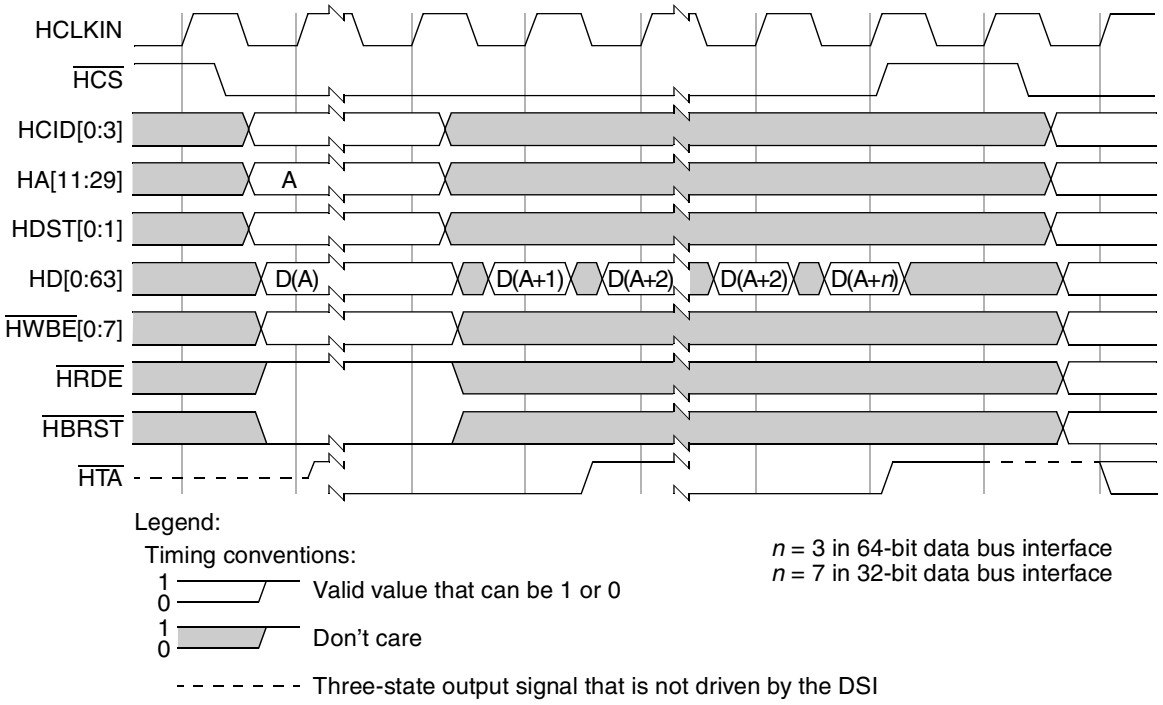


Figure 10-84. Synchronous Burst Write to MSC8122 DSI

The DSI samples HA, HDST, HCID, HD, HWBE, HRDE, and HBRST on the first HCLKIN rising edge on which HCS is asserted. If HCID[0:3] match the CHIPID value, the DSI is accessed. HWBE are asserted, HBRST is asserted, and HRDE is negated. Assertion of HTA indicates that the DSI is ready to complete the current beat of the access and the host must proceed to the next beat of this access. When the host reaches the last beat of the access, it must terminate the burst access. Typically HTA is asserted immediately for each beat of the access. If the write buffer is full, HTA assertion is delayed. Because HTA is connected to the LUPWAIT signal of the UPM, all local bus signals are frozen until HTA goes to 0 and then the UPM continues in its pattern. After the last beat of the access, HTA is driven to logic 1 and stops being driven on the next rising edge of HCLKIN. The host can start its next access to the same MSC8122 immediately in the next HCLKIN rising edge without negating HCS between accesses. If the next access is not to the same MSC8122, then, to prevent contention on HTA, the host must wait to access the next device until the previous DSI stops driving HTA. The easiest way to achieve this is to insert idle cycles at the end of the UPM pattern to guarantee that HTA is inactive.

10.5.6.3.4 Synchronous Burst Read

Figure 10-85 shows a synchronous burst read access. The DSI samples HA, HDST, HCID, $\overline{\text{HWBE}}$, $\overline{\text{HRDE}}$, and $\overline{\text{HBRST}}$ on the first HCLKIN rising edge on which $\overline{\text{HCS}}$ is asserted. If HCID[0:3] match the CHIPID value, the DSI is accessed.

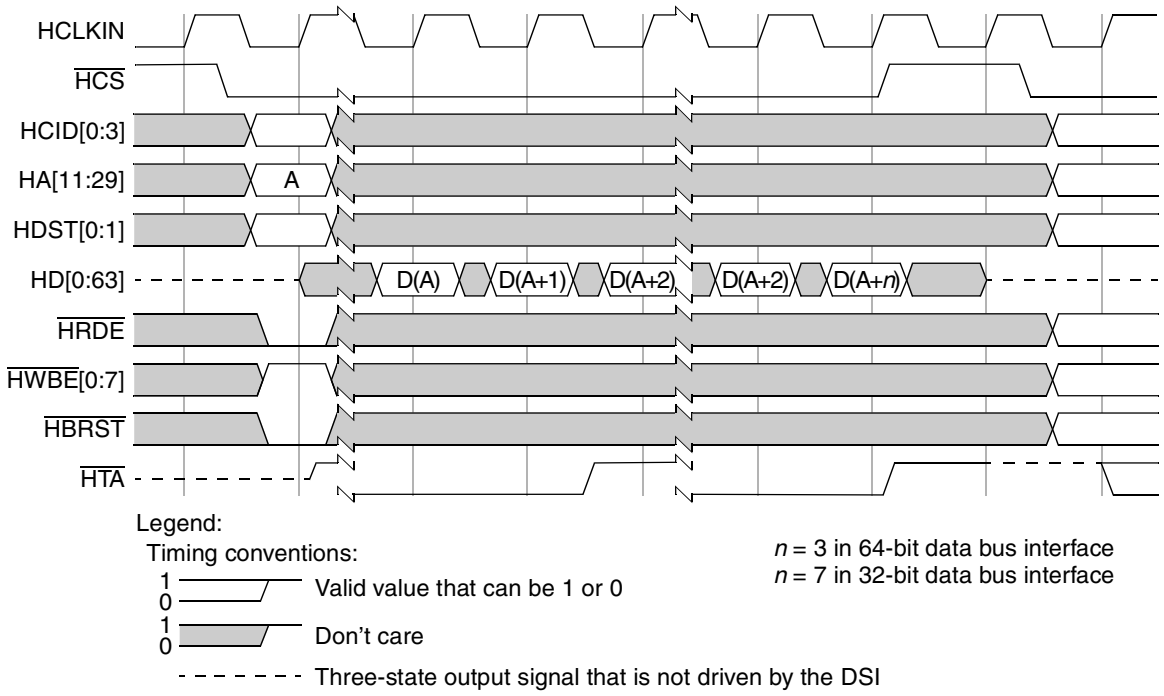


Figure 10-85. Synchronous Burst Read from MSC8122 DSI

$\overline{\text{HRDE}}$ and $\overline{\text{HBRST}}$ are asserted and $\overline{\text{HWBE}}$ are negated. When DCR[RPE] (see the MSC8122 documentation) is set, a burst read access initiates data prefetching from consecutive addresses in the internal memory space. Assertion of $\overline{\text{HTA}}$ indicates that data is valid for the current beat of the access and the host must proceed to the next beat of this access. Because $\overline{\text{HTA}}$ is connected to the LUPWAIT signal of the UPM, all local bus signals are frozen until $\overline{\text{HTA}}$ goes to 0 and then the UPM continues in its pattern. When the host reaches the last beat of the access, it must terminate the burst access. The $\overline{\text{HTA}}$ is asserted earlier when the data for this access is already prefetched to the read buffer. Typically, after the first beat of the burst access, $\overline{\text{HTA}}$ remains asserted until the end of the access. After the last beat of the access, $\overline{\text{HTA}}$ is driven to 1 and stops being driven in the next rising edge of HCLKIN. The host can start its next access to the same MSC8122 immediately in the next HCLKIN rising edge without negating $\overline{\text{HCS}}$ between accesses. If the next access is not to the same MSC8122, to prevent contention on $\overline{\text{HTA}}$, the host must wait to access the next device until the previous DSI stops driving $\overline{\text{HTA}}$. The easiest way to achieve this is insert idle cycles at the end of the UPM pattern to guarantee that $\overline{\text{HTA}}$ is inactive.

10.5.6.4 Broadcast Accesses

By using $\overline{\text{HBCS}}$, a host can share one chip-select signal between multiple MSC8122 devices for broadcasting write accesses. In broadcast mode, the DSI does not drive its $\overline{\text{HTA}}$ signal to prevent contention between multiple devices driving different values to the same signal. Also, the DSI does not decode HCID[0:3].

Note that broadcasting is allowed only for write accesses.

The DSI sets the overflow bit, DER[OVF], if there is an overflow during broadcast accesses. This bit can be cleared by writing a value of 1 to it. To avoid overflow when accessing DSI registers during broadcast accesses, wait at least 10 host clock cycles in synchronous mode or 8 internal clock cycles in asynchronous mode between each DSI register access.

To avoid data corruption, if DER[OVF] is set, no broadcast access is written until the bit is reset. Therefore, after the last broadcast access, and before any regular write access, DER[OVF] must first be read and reset if it is set.

NOTE

In asynchronous mode, write data from a previous access (even a normal write access) may be lost due to overflow during broadcast accesses. To prevent such a loss, ensure that previous access data has propagated to the FIFO or DSI registers, depending on the type of previous access. This can be achieved by performing a read access before the first broadcast access.

In broadcast accesses, the host must comply with the following rules:

- In asynchronous mode, $\overline{\text{HWBS}}$[0:3]/$\overline{\text{HDBS}}$[0:3] assertion time should be at least the minimum, which is defined in the AC characteristics section of the *MSC8122 Technical Data* sheet.
- In synchronous mode single access, the host must wait 1 cycle before terminating the access. Access signals must be in the same valid state during two positive edges of the host clock cycles. Access duration is two clock cycles (the DSI may translate accesses lasting longer than two clock cycles as two or more back-to-back accesses).
- In synchronous mode burst accesses, broadcast accesses are not allowed.

Chapter 11
Sequencer

11.1 Overview

The I/O sequencer switches transactions among its ports, using a buffer pool to minimize blocking. Figure 11-1 is a block diagram of the I/O sequencer (IOS).

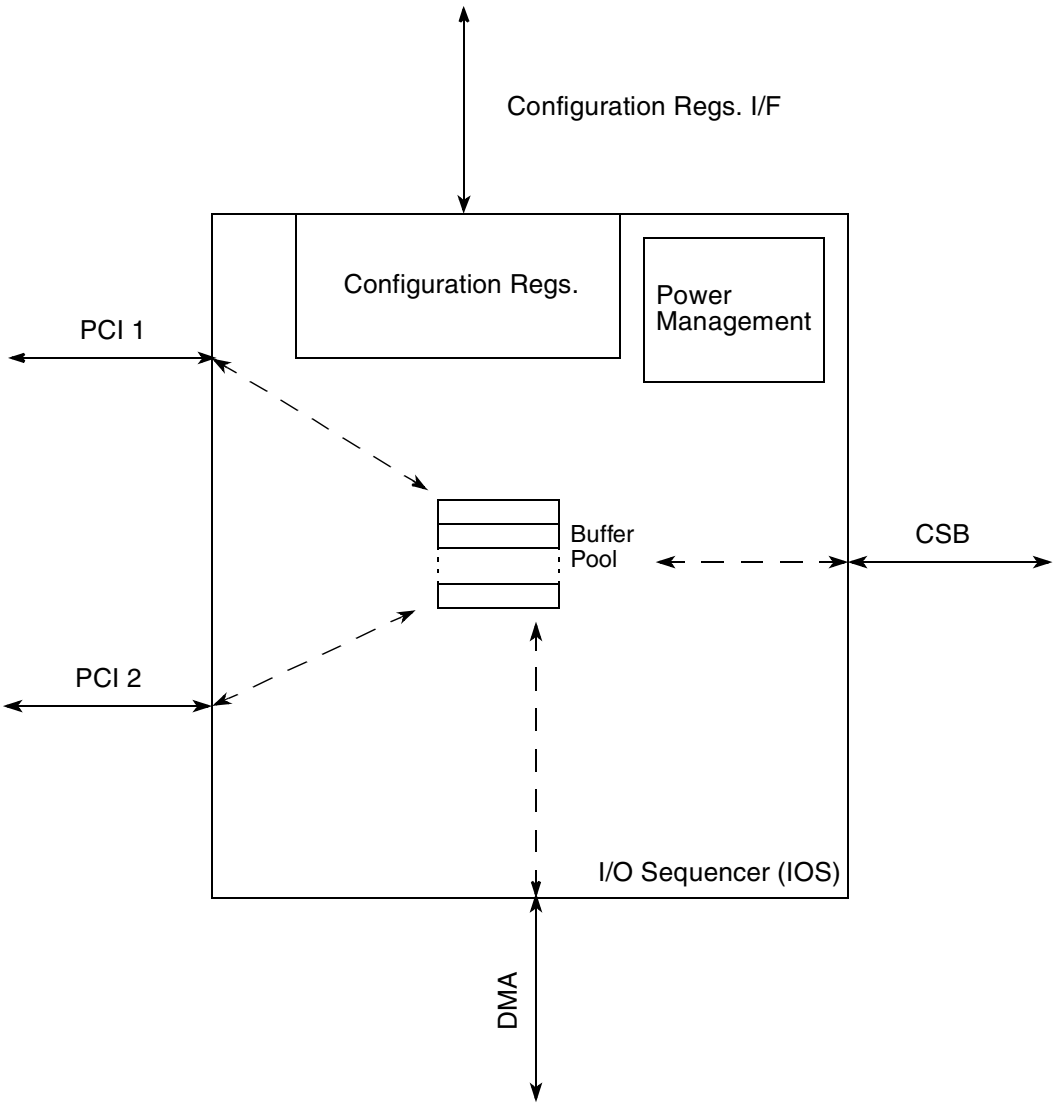


Figure 11-1. I/O Sequencer Block Diagram

11.1.1 Features

The I/O sequencer includes the following features:

- Switches transactions among its ports
- Contains 12 cache-line (32 byte) buffers to allow streaming of PCI transactions
- Performs address translation on outbound PCI transactions

Note that the number of CSB masters allowed to access to PCI outbound windows is restricted to no more than four non-CPU masters or no more than two non-CPU plus the CPU. If this number is exceeded, deadlock can occur on CSB arbitration.

11.2 External Signal Description

The I/O sequencer has no external signals.

11.3 Memory Map/Register Definition

Table 11-1 shows the I/O sequencer memory map.

Table 11-1. Sequencer Memory Map

Offset	Register	Access	Reset	Section/Page
0x00	POTAR0—PCI outbound translation address register 0	R/W	0x0000_0000	11.4.1/11-3
0x08	POBAR0—PCI outbound base address register 0	R/W	0x0000_0000	11.4.2/11-3
0x10	POCMR0—PCI outbound comparison mask register 0	R/W	0x0000_0000	11.4.3/11-4
0x18	POTAR1—PCI outbound translation address register 1	R/W	0x0000_0000	11.4.1/11-3
0x20	POBAR1—PCI outbound base address register 1	R/W	0x0000_0000	11.4.2/11-3
0x28	POCMR1—PCI outbound comparison mask register 1	R/W	0x0000_0000	11.4.3/11-4
0x30	POTAR2—PCI outbound translation address register 2	R/W	0x0000_0000	11.4.1/11-3
0x38	POBAR2—PCI outbound base address register 2	R/W	0x0000_0000	11.4.2/11-3
0x40	POCMR2—PCI outbound comparison mask register 2	R/W	0x0000_0000	11.4.3/11-4
0x48	POTAR3—PCI outbound translation address register 3	R/W	0x0000_0000	11.4.1/11-3
0x50	POBAR3—PCI outbound base address register 3	R/W	0x0000_0000	11.4.2/11-3
0x58	POCMR3—PCI outbound comparison mask register 3	R/W	0x0000_0000	11.4.3/11-4
0x60	POTAR4—PCI outbound translation address register 4	R/W	0x0000_0000	11.4.1/11-3
0x68	POBAR4—PCI outbound base address register 4	R/W	0x0000_0000	11.4.2/11-3
0x70	POCMR4—PCI outbound comparison mask register 4	R/W	0x0000_0000	11.4.3/11-4
0x78	POTAR5—PCI outbound translation address register 5	R/W	0x0000_0000	11.4.1/11-3
0x80	POBAR5—PCI outbound base address register 5	R/W	0x0000_0000	11.4.2/11-3
0x88	POCMR5—PCI outbound comparison mask register 5	R/W	0x0000_0000	11.4.3/11-4

Table 11-1. Sequencer Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0xF0	PMCR—Power management control register	R/W	0x0000_0000	11.4.4/11-5
0xF8	DTCR—Discard timer control register	R/W	0x0000_0000	11.4.5/11-6

11.4 Register Descriptions

11.4.1 PCI Outbound Translation Address Registers (POTAR*n*)

The PCI outbound translation address register defines the location of the outbound translation window in the PCI (translated) address space. Figure 11-2 shows the POTAR*n* register fields.

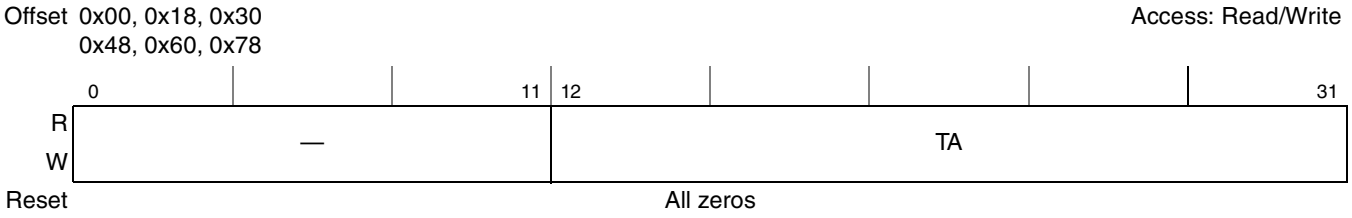


Figure 11-2. PCI Outbound Translation Address Registers (POTAR*n*)

Table 11-2 describes POTAR*n* fields.

Table 11-2. POTAR*n* Field Descriptions

Bits	Name	Description
0–11	—	Reserved
12–31	TA	Translation address. Contains the starting address of the outbound translated address. It also corresponds to the most-significant 20 bits of a 32-bit address.

11.4.2 PCI Outbound Base Address Registers (POBAR*n*)

The PCI outbound base address register (POBAR*n*) defines the location of the outbound translation window in the local (source) memory space. Figure 11-3 shows the POBAR*n* register fields.

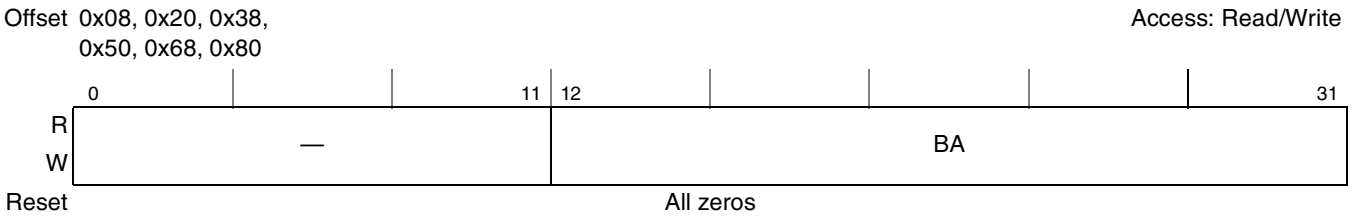


Figure 11-3. PCI Outbound Base Address Registers (POBAR*n*)

Table 11-3 describes POBAR*n* fields.

Table 11-3. POBAR*n* Field Descriptions

Bits	Name	Description
0–11	—	Reserved
12–31	BA	Base address. This field contains the starting address of the outbound translated window. This field corresponds to the most-significant 20 bits of a 32-bit address.

11.4.3 PCI Outbound Comparison Mask Registers (POCMR*n*)

The PCI outbound comparison mask register (POCMRn) defines the size and destination of the outbound translation window. It also defines some properties of the window in the PCI address space. See Section 11.5.1, "Transaction Forwarding," for more information. Figure 11-4 shows the POCMR*n* register fields.

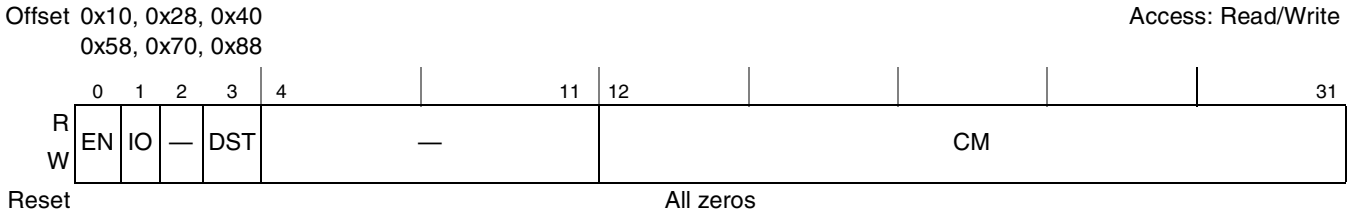


Figure 11-4. PCI Outbound Comparison Mask Registers (POCMR*n*)

Table 11-4 describes the bit settings of the POCMR*n* register.

Table 11-4. POCMR*n* Field Descriptions

Bits	Name	Description
0	EN	Enable. Enables the address translation window. 0 Address translation is disabled for this window. 1 Address translation is enabled for this window. Local addresses that match the definition of the window will be recognized by the device and translated to the PCI memory space.
1	IO	I/O space. Determines whether the window is mapped to the PCI memory space or PCI I/O space. 0 Memory space 1 I/O space
2	—	Reserved
3	DST	Destination. Determines the interface on which transactions that are translated through this window will be performed. 0 PCI 1 1 PCI 2
4–11	—	Reserved, should be cleared.

Table 11-4. POCMR*n* Field Descriptions (continued)

Bits	Name	Description
12–31	CM	Comparison mask. Contains the size of the translation window. The bits that are 1 in this field indicate bits of the transaction address that should be matched to the value in the PCI outbound base address register and translated in case of a match. This field corresponds to the most-significant 20 bits of a 32-bit address.

Value	Meaning	Value	Meaning
0000_0000_0000_0000_0000	4 GB	1111_1111_1110_0000_0000	2 MB
1000_0000_0000_0000_0000	2 GB	1111_1111_1111_0000_0000	1 MB
1100_0000_0000_0000_0000	1 GB	1111_1111_1111_1000_0000	512 KB
1110_0000_0000_0000_0000	512 MB	1111_1111_1111_1100_0000	256 KB
1111_0000_0000_0000_0000	256 MB	1111_1111_1111_1110_0000	128 KB
1111_1000_0000_0000_0000	128 MB	1111_1111_1111_1111_0000	64 KB
1111_1100_0000_0000_0000	64 MB	1111_1111_1111_1111_1000	32 KB
1111_1110_0000_0000_0000	32 MB	1111_1111_1111_1111_1100	16 KB
1111_1111_0000_0000_0000	16 MB	1111_1111_1111_1111_1110	8 KB
1111_1111_1000_0000_0000	8 MB	1111_1111_1111_1111_1111	4 KB
1111_1111_1100_0000_0000	4 MB		

All others values are Reserved

11.4.4 Power Management Control Register (PMCR)

PMCR provides control of system-level low-power modes. Figure 11-5 shows the PMCR register fields.

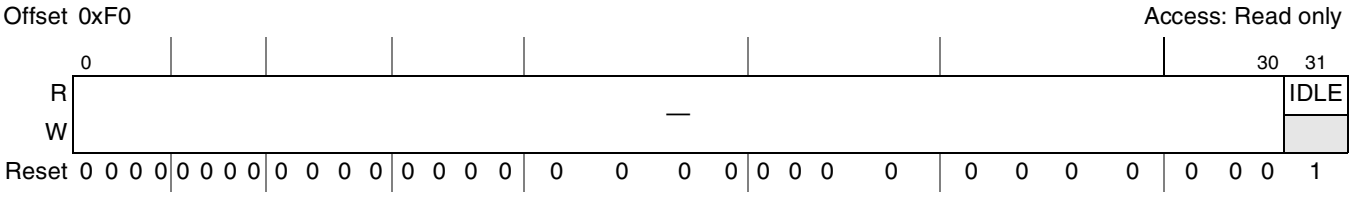


Figure 11-5. Power Management Control Register (PMCR)

Table 11-5 describes PMCR fields.

Table 11-5. PMCR Field Descriptions

Bits	Name	Description
0–30	—	Reserved
31	IDLE	This read-only bit indicates that the logic controlled by the <<BLOCK NAME>> power management function is idle. This bit could be used as an indication that it is okay to disable the clocks to this complex. 0 Logic is active. 1 Logic is idle. There are no outstanding transactions in the IOS, the DMA, or either of the PCI ports.

11.4.5 Discard Timer Control Register (DTCR)

DTCR configures the discard timer, which is used to place a time limit on PCI delayed read transactions from non-prefetchable memory. Although prefetched reads may be discarded whenever the IOS is full and needs to allocate another buffer, other delayed reads must not be discarded until the originator actually receives the data. The DTCR is used to release stuck buffers in case of malfunctioning or disconnected masters that never come back to read the data they requested

Figure 11-6 shows the DTCR register fields.

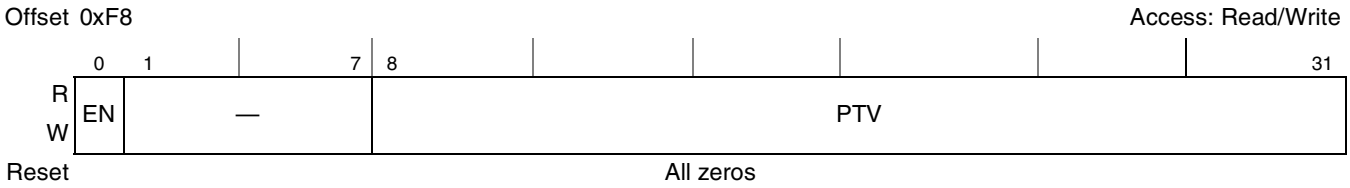


Figure 11-6. Discard Timer Control Register (DTCR)

Table 11-6 describes DTCR fields.

Table 11-6. DTCR Field Descriptions

Bits	Name	Description
0	EN	Enable. This bit enables the discard timer. 0 Disabled 1 Enabled
1–7	—	Reserved
8–31	PTV	Preload timer value (PTV). This field contains the preload value for the discard timer. PCI delayed reads from non-prefetchable address space are discarded after (2^{24} – PTV) internal clock cycles if the master has not repeated the transaction. 0xFFFFFF is not valid for PTV. For example, to discard a delayed completion if the PCI master has not repeated the transaction in 2^{15} PCI clocks, —assuming the internal frequency is twice the PCI frequency —the PTV should equal $2^{24} – 2^{16}$ (0xFF0000).

11.5 Functional Description

The IOS is a four-port switch with buffering. Each port has master and slave interfaces. When a port masters a transaction, the transaction attributes are stored in a buffer and the IOS generates a transaction to the slave interface of the destination port. The data is also buffered between the ports. The IOS contains 12 cache line (32-byte) transaction buffers, some of which are reserved for specific types of transactions.

The address and data phases of the transactions are independent. The data phases of the transactions are not required to be inorder.

11.5.1 Transaction Forwarding

Although the ports use a similar interface, the I/O sequencer is not actually symmetrical. The transaction forwarding from each source is explained in the following sections.

11.5.1.1 Transactions from the Coherency System Bus (CSB) Port

Transactions from the CSB port are forwarded as follows:

- If the address matches the 12-byte PCI controller software configuration memory space of either PCI controller, the transaction is forwarded to PCI 1 or PCI 2, according to the match. These address values are configuration options of the I/O sequencer. See Table 13-3 for more information on PCI controller software configuration memory space.
- If the address matches the DMA register memory space, the transaction is forwarded to the DMA port.
- If the address hits any of the outbound translation windows, the transaction is forwarded to PCI 1 or PCI 2, according to POCMR[DST], with the address translated. See Section 11.5.2, "PCI Outbound Address Translation," for more information.
- When booting from a PCI port, transactions that do not hit any of the above are forwarded to PCI 1 or PCI 2, according to the boot configuration specified in the ROMLOC field of the reset configuration word high register described Section 4.3.2.2, "Reset Configuration Word High Register (RCWHR)."

11.5.1.2 Transactions from the PCI Ports

Transactions from the PCI ports are forwarded as follows:

- Transaction from one PCI port to another PCI port is not supported.
- If the address matches the DMA register memory space, the transaction is forwarded to the DMA port.
- All other transactions are forwarded to the CSB port.

11.5.1.3 Transactions from the DMA Port

Transactions from the DMA port are forwarded as follows:

- If the address hits any of the outbound translation windows, the transaction is forwarded to PCI1 or PCI 2, according to the DST bit of the POCMR, with the address translated. See Section 11.5.2, "PCI Outbound Address Translation," for more information.
- All other transactions are forwarded to the CSB port.

11.5.2 PCI Outbound Address Translation

Outbound address translation is provided to allow the outbound transactions to access any address over the PCI memory or I/O space. Translation window base addresses are defined in the PCI outbound base address registers. See Section 11.4.2, "PCI Outbound Base Address Registers (POBARn)," for more information. Transactions to these address ranges are issued on the PCI bus with a translated address. The translation addresses are defined in the associated PCI outbound translation address registers (POTARs).

Figure 11-7 shows an example translation window for outbound memory accesses.

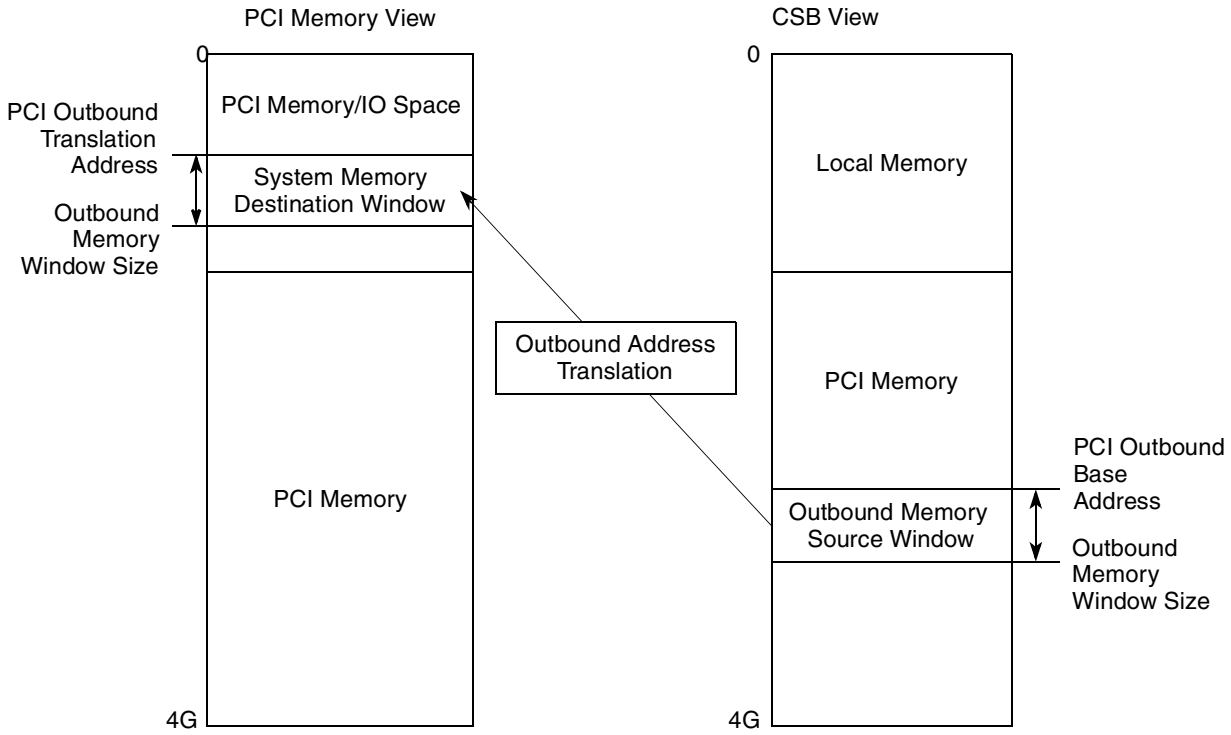


Figure 11-7. Outbound PCI Memory Address Translation

The six sets of outbound translation registers allow six simultaneous translation windows to the PCI ports. Each window is programmed for one of the PCI ports. Software can move and adjust the memory window translations and sizes during run-time. This allows software to access different PCI memory/IO spaces on the fly, but the PCI outbound translation source windows must not overlap. However, outbound translation destination windows can be overlapped.

11.5.3 Transaction Ordering

The following rules are applied to maintain proper ordering of transactions:

- The transactions arriving from each port are dispatched to the destination port in the order of arrival. The dispatch order of transactions arriving on different ports is not necessarily maintained.
- A read transaction that originates at the CSB port and reads from a PCI port pulls out of the IOS any posted writes that originated on the PCI port and were posted before the read data arrives from the PCI.
- The IOS can always accept a write from a PCI port without forcing the PCI port to first accept a read.

Chapter 12
DMA/Messaging Unit

The DMA/messaging unit supports communication between two processors on different buses, for example, a local processor and a processor on a PCI bus. This unit operates with generic messages and doorbell registers. Figure 12-1 is a block diagram of the DMA/messaging unit.

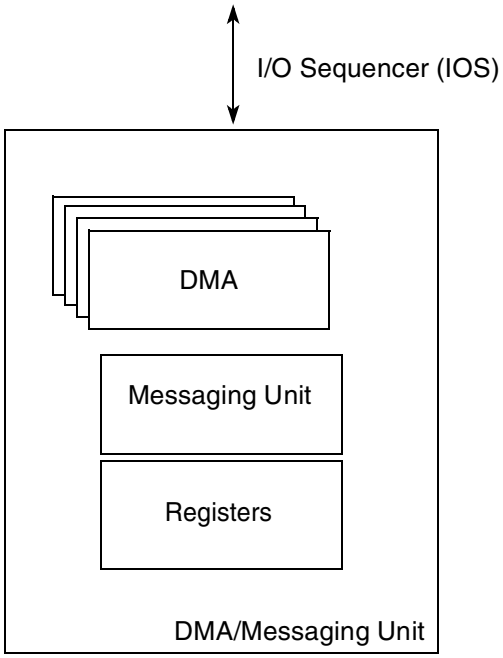


Figure 12-1. DMA/Messaging Unit Block Diagram

This block also provides a DMA controller, which transfers blocks of data independent of the local processor or PCI hosts. The DMA module has four high-speed DMA channels, which share buffer space in the I/O sequencer (IOS) to facilitate the gathering and sending of data.

12.1 Features

The DMA/messaging unit includes the following features:

- Message and doorbell registers for inter-processor communication
- DMA controller
  - — Four DMA channels
  - — Concurrent execution across multiple channels with programmable bandwidth control
  - — Misaligned transfer capability
  - — Data chaining and direct mode
  - — Interrupt on completed segment, chain, and error
  - — Optional external control signals (REQ/ACK/DONE) per channel

12.2 External Signal Description

This section describes the DMA signals.

12.2.1 Detailed Signal Descriptions

Table 12-1 contains the detailed descriptions of the DMA interface signals.

Table 12-1. DMA Interface Signals—Detailed Signal Descriptions

Signal	I/O	Description	
$\overline{\text{DREQ}}$[0:3]	I	DMA request signals, one per channel. The DMA request signal indicates the start or continuation of a DMA transfer. The falling edge of $\overline{\text{DREQ}}n$ causes DMAMR*n*[CS] to be set, thereby activating the DMA channel.	
		State Meaning	Asserted—Assertion of $\overline{\text{DREQ}}n$ starts or resumes a DMA transfer if DMAMR*n*[EMSEN] is 1. Negated—Negation of $\overline{\text{DREQ}}n$ has no effect.
		Timing	Assertion—Can be asserted asynchronously. Negation—Should remain asserted until $\overline{\text{DACK}}n$ is asserted or the requested transaction to the peripheral occurs.
$\overline{\text{DACK}}$[0:3]	O	DMA acknowledge signals, one per channel. The DMA acknowledge signal reflects the value of DMAMR*n*[CS].	
		State Meaning	Asserted—A DMA transfer is active. Negated—The DMA transfer is halted or complete.
		Timing	Assertion—Asserted asynchronously when a DMA transfer is started or resumed in the internal control logic. Negation—Negated asynchronously when a DMA transfer is halted or completed in the internal control logic. Note that there may still be outstanding write transactions in the bus pipeline after the negation of $\overline{\text{DACK}}n$.

Table 12-1. DMA Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
$\overline{\text{DDONE}}$[0:3]	O	DMA done signals, one per channel. The DMA done signal indicates that the DMA transfer has completed.	
		State Meaning	Asserted—A DMA transfer is complete. Negated—A DMA transfer is active or halted.
		Timing	Assertion—Asserted asynchronously when a DMA transfer is completed in the internal control logic. Note that there may still be outstanding write transactions in the bus pipeline after the assertion of $\overline{\text{DDONE}}n$. Negation—Negated asynchronously when a DMA transfer begins in the internal control logic.

12.3 Memory Map/Register Definition

Table 12-2 lists the address and access of the memory map module.

Table 12-2. Module Memory Map

Offset	Register	Access	Reset	Section/Page
0x0_8030	OMISR—Outbound message interrupt status register	special	0x0000_0000	12.4.1/12-4
0x0_8034	OMIMR—Outbound message interrupt mask register	R/W	0x0000_0000	12.4.2/12-6
0x0_8050	IMR0—Inbound message register 0	R/W	0x0000_0000	12.4.3/12-7
0x0_8054	IMR1—Inbound message register 1	R/W	0x0000_0000	12.4.3/12-7
0x0_8058	OMR0—Outbound message register 0	R/W	0x0000_0000	12.4.4/12-7
0x0_805C	OMR1—Outbound message register 1	R/W	0x0000_0000	12.4.4/12-7
0x0_8060	ODR—Outbound doorbell register	R/W	0x0000_0000	12.4.5/12-8
0x0_8068	IDR—Inbound doorbell register	R/W	0x0000_0000	12.4.5/12-8
0x0_8080	IMISR—Inbound message interrupt status register	R/W	0x0000_0000	12.4.6/12-9
0x0_8084	IMIMR—Inbound message interrupt mask register	R/W	0x0000_0000	12.4.7/12-11
0x0_8100	DMAMR0—DMA 0 mode register	R/W	0x0000_0000	12.4.8.1/12-12
0x0_8104	DMASR0—DMA 0 status register	R/W	0x0000_0000	12.4.8.2/12-14
0x0_8108	DMACDAR0—DMA 0 current descriptor address register	R/W	0x0000_0000	12.4.8.3/12-15
0x0_8110	DMASAR0—DMA 0 source address register	R/W	0x0000_0000	12.4.8.4/12-16
0x0_8118	DMADAR0—DMA 0 destination address register	R/W	0x0000_0000	12.4.8.5/12-16
0x0_8120	DMABCR0—DMA 0 byte count register	R/W	0x0000_0000	12.4.8.6/12-17
0x0_8124	DMANDAR0—DMA 0 next descriptor address register	R/W	0x0000_0000	12.4.8.7/12-17
0x0_8180	DMAMR1—DMA 1 mode register	R/W	0x0000_0000	12.4.8.1/12-12
0x0_8184	DMASR1—DMA 1 status register	R/W	0x0000_0000	12.4.8.2/12-14
0x0_8188	DMACDAR1—DMA 1 current descriptor address register	R/W	0x0000_0000	12.4.8.3/12-15
0x0_8190	DMASAR1—DMA 1 source address register	R/W	0x0000_0000	12.4.8.4/12-16

Table 12-2. Module Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x0_8198	DMADAR1—DMA 1 destination address register	R/W	0x0000_0000	12.4.8.5/12-16
0x0_81A0	DMABCR1—DMA 1 byte count register	R/W	0x0000_0000	12.4.8.6/12-17
0x0_81A4	DMANDAR1—DMA 1 next descriptor address register	R/W	0x0000_0000	12.4.8.7/12-17
0x0_8200	DMAMR2—DMA 2 mode register	R/W	0x0000_0000	12.4.8.1/12-12
0x0_8204	DMASR2—DMA 2 status register	R/W	0x0000_0000	12.4.8.2/12-14
0x0_8208	DMACDAR2—DMA 2 current descriptor address register	R/W	0x0000_0000	12.4.8.3/12-15
0x0_8210	DMASAR2—DMA 2 source address register	R/W	0x0000_0000	12.4.8.4/12-16
0x0_8218	DMADAR2—DMA 2 destination address register	R/W	0x0000_0000	12.4.8.5/12-16
0x0_8220	DMABCR2—DMA 2 byte count register	R/W	0x0000_0000	12.4.8.6/12-17
0x0_8224	DMANDAR2—DMA 2 next descriptor address register	R/W	0x0000_0000	12.4.8.7/12-17
0x0_8280	DMAMR3—DMA 3 mode register	R/W	0x0000_0000	12.4.8.1/12-12
0x0_8284	DMASR3—DMA 3 status register	R/W	0x0000_0000	12.4.8.2/12-14
0x0_8288	DMACDAR3—DMA 3 current descriptor address register	R/W	0x0000_0000	12.4.8.3/12-15
0x0_8290	DMASAR3—DMA 3 source address register	R/W	0x0000_0000	12.4.8.4/12-16
0x0_8298	DMADAR3—DMA 3 destination address register	R/W	0x0000_0000	12.4.8.5/12-16
0x0_82A0	DMABCR3—DMA 3 byte count register	R/W	0x0000_0000	12.4.8.6/12-17
0x0_82A4	DMANDAR3—DMA 3 next descriptor address register	R/W	0x0000_0000	12.4.8.7/12-17
0x0_82A8	DMAGSR—DMA general status register	R	0x0000_0000	12.4.8.8/12-18
0x0_82B0–0x0_82FF	Reserved	—	—	—

12.4 Register Descriptions

The following sections describe the DMA/messaging unit configuration, control, and status registers. Note that any messaging unit register that is documented as being accessible from the PCI bus can be accessed from either PCI bus. However, there is only one $\overline{\text{INTA}}$ signal, so it is recommended to use the outbound portion of the messaging unit with the PCI bus that uses the $\overline{\text{INTA}}$ signal.

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.

12.4.1 Outbound Message Interrupt Status Register (OMISR)

OMISR contains the interrupt status of the doorbell and outbound message registers. A PCI device acknowledges the outbound message interrupt by writing a 1 to the appropriate status bit: OMISR[OM1I]

or OMISR[OM0I]. Setting one of these bits clears both the interrupt and the corresponding status bit. The local processor provokes an outbound message interrupt by writing to either of the two outbound message registers: OMR0 or OMR1. OMISR can be accessed from the CSB or the PCI bus, but it is normally accessed only from the PCI bus. Figure 12-2 shows the OMISR fields.

Offset: 0x030 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	All zeros															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	ODI	0	OM1I	OM0I
W																
Reset	All zeros															

Figure 12-2. Outbound Message Interrupt Status Register (OMISR)

Table 12-3 describes the OMISR register.

Table 12-3. OMISR Field Descriptions

Bits	Name	Description
31–4	—	Reserved
3	ODI	Outbound doorbell interrupt. This read-only bit indicates the status of the ODR bits. It is masked by OMIMR[ODIM]. 0 No outbound doorbell interrupt. 1 There is an outbound doorbell interrupt.
2	—	Reserved
1	OM1I	Outbound message 1 interrupt. When set, indicates that there is an outbound message 1 interrupt. Write 1 to this position to clear this bit. 0 No outbound message 1 interrupt. 1 There is an outbound message 1 interrupt.
0	OM0I	Outbound message 0 interrupt. When set, indicates that there is an outbound message 0 interrupt. Write 1 to this position to clear this bit. 0 No outbound message 0 interrupt. 1 There is an outbound message 0 interrupt.

12.4.2 Outbound Message Interrupt Mask Register (OMIMR)

OMIMR contains the interrupt mask of the doorbell and message register events generated by the local processor. OMIMR can be read from the CSB or the PCI bus, but it can be cleared only from the PCI bus. Figure 12-3 shows the OMIMR.

Offset: 0x034 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	All zeros															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	ODIM	0	OM1IM	OM0IM
W																
Reset	All zeros															

Figure 12-3. Outbound Message Interrupt Mask Register (OMIMR)

Table 12-4 describes the OMIMR register.

Table 12-4. OMIMR Field Descriptions

Bits	Name	Description
31–4	—	Reserved
3	ODIM	Outbound doorbell interrupt mask. 0 Outbound doorbell interrupt is allowed. 1 Outbound doorbell interrupt is masked.
2	—	Reserved
1	OM1IM	Outbound message 1 interrupt mask. 0 Outbound message 1 interrupt is allowed. 1 Outbound message 1 interrupt is masked.
0	OM0IM	Outbound message 0 interrupt mask. 0 Outbound message 0 interrupt is allowed. 1 Outbound message 0 interrupt is masked.

12.4.3 Inbound Message Registers

The inbound message registers can be read from the PCI bus and the CSB in both host and agent modes. They can be written only from the PCI bus. Figure 12-4 shows the IMR0 and IMR1 fields.

Figure 12-4. Inbound Message Registers (IMR0, IMR1)

Table 12-5 describes the IMR*n* register.

Table 12-5. IMR0 and IMR1 Field Descriptions

Bits	Name	Description
31–0	IMSG*n*	Inbound message *n*. Contains generic data to be passed between the local processor and external hosts.

12.4.4 Outbound Message Registers (OMR0–OMR1)

The outbound message registers can be read from the PCI bus and the CSB in both host and agent modes. They can be written only from the CSB.

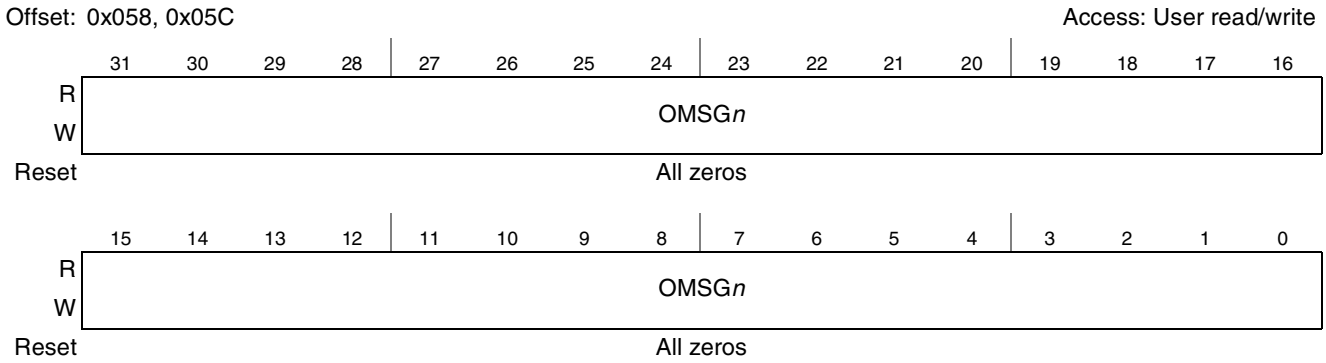


Figure 12-5. Outbound Message Registers (OMR0–OMR1)

Table 12-6 describes the OMR*n* registers.

Table 12-6. OMR0 and OMR1 Field Descriptions

Bits	Name	Description
31–0	OMSG*n*	Outbound message *n*. Contains generic data to be passed between the local processor and external hosts.

12.4.5 Doorbell Registers

The following sections describe the outbound and inbound doorbell registers.

12.4.5.1 Outbound Doorbell Register (ODR)

ODR is accessible from the PCI bus and the CSB in both host and agent modes. Figure 12-6 shows the ODR*n* fields.

Offset: 0x060 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	ODR 28	ODR 27	ODR 26	ODR 25	ODR 24	ODR 23	ODR 22	ODR 21	ODR 20	ODR 19	ODR 18	ODR 17	ODR 16
W																
Reset	All zeros															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ODR 15	ODR 14	ODR 13	ODR 12	ODR 11	ODR 10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
W																
Reset	All zeros															

Figure 12-6. Outbound Doorbell Register (ODR)

Table 12-7 describes the ODR registers.

Table 12-7. ODR Field Descriptions

Bits	Name	Description
31–29	—	Reserved
28–0	ODR*n*	Outbound doorbell *n.* Write 1 from the CSB to set. Write 1 from the PCI bus to clear. Writing 0 has no effect. (Writing a bit in this register from the CSB causes an interrupt ($\overline{\text{INTA}}$) to be generated.)

12.4.5.2 Inbound Doorbell Register (IDR)

IDR is accessible from the PCI bus and the CSB in both host and agent modes. Figure 12-7 shows the IDR fields.

Offset: 0x068 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	IMC	IDR30	IDR29	IDR28	IDR27	IDR26	IDR25	IDR24	IDR23	IDR22	IDR21	IDR20	IDR19	IDR18	IDR17	IDR16
Reset	All zeros															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
Reset	All zeros															

Figure 12-7. Inbound Doorbell Register (IDR)

Table 12-8 describes the IDR registers.

Table 12-8. IDR Field Descriptions

Bits	Name	Descriptions
31	IMC	Inbound machine check. Write 1 from the PCI bus to set. Write 1 from the CSB to clear. Writing 0 has no effect. Writing this bit from the PCI bus causes a machine check interrupt to be generated to the local processor.
30–0	IDR*n*	Inbound doorbell *n*. Write 1 from the PCI bus to set. Write 1 from the CSB to clear. Writing 0 has no effect. Writing a bit in this register from the PCI bus causes an interrupt to be generated to the local processor.

12.4.6 Inbound Message Interrupt Status Register (IMISR)

The IMISR contains the interrupt status of the doorbell and message register events. Writing a 1 to IM1I clears the bit. The events are generated by the PCI masters. IMISR should be accessed only from the CSB and only in agent mode. Accesses while in host mode or from the PCI bus have undefined results.

Figure 12-8 shows the IMISR fields.

Offset: 0x080 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	All zeros															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	MCI	IDI	0	IM1I	IM0I
W																
Reset	All zeros															

Figure 12-8. Inbound Message Interrupt Status Register (IMISR)

Table 12-9 describes the IMISR register.

Table 12-9. IMISR Field Descriptions

Bits	Name	Descriptions
31–5	—	Reserved
4	MCI	Machine check interrupt. Indicates whether a machine check interrupt condition was generated by setting the IDR[31]. The interrupt is cleared by writing a 1 to IDR[IMC] from the CSB. 0 No machine check interrupt. 1 There is a machine check interrupt.
3	IDI	Inbound doorbell interrupt. Indicates whether an inbound doorbell interrupt occurred. 0 No inbound doorbell interrupt. 1 There is an inbound doorbell interrupt.
2	—	Reserved
1	IM1I	Inbound message 1 interrupt. Indicates whether an inbound message 1 interrupt occurred. Write 1 to this position to clear this bit. 0 No Inbound Message 1 interrupt. 1 There is an Inbound Message 1 interrupt.
0	IM0I	Inbound message 0 interrupt. Indicates whether an inbound message 0 interrupt occurred. Write 1 to this position to clear this bit. 0 No inbound message 0 interrupt. 1 There is an inbound message 0 interrupt.

12.4.7 Inbound Message Interrupt Mask Register (IMIMR)

This register contains the interrupt mask of the doorbell and message register events generated by the PCI master. IMIMR should be accessed only from the CSB and only in agent mode. Accesses while in host mode or from the PCI bus have undefined results. Figure 12-9 shows the IMIMR fields.

Offset: 0x084 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	All zeros															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	MCIM	IDIM	0	IM1IM	IM0IM
W																
Reset	All zeros															

Figure 12-9. Inbound Message Interrupt Mask Register (IMIMR)

Table 12-10 describes the IMISR register.

Table 12-10. IMIMR Field Descriptions

Bits	Name	Description
31–5	—	Reserved
4	MCIM	Machine check interrupt mask. 0 Machine check interrupt from the IDR is allowed. 1 Machine check interrupt is masked. IMISR[MC1] is cleared.
3	IDIM	Inbound doorbell interrupt mask. 0 Inbound doorbell interrupt is allowed. 1 Inbound doorbell interrupt is masked. IMISR[IDI] is cleared.
2	—	Reserved
1	IM1IM	Inbound message 1 interrupt mask. 0 Inbound message 1 interrupt is allowed. 1 Inbound message 1 interrupt is masked. IMISR[IM1I] is cleared.
0	IM0IM	Inbound message 0 interrupt mask. 0 Inbound message 0 interrupt is allowed. 1 Inbound message 0 interrupt is masked. IMISR[IM0I] is cleared.

12.4.8 DMA Registers

Each DMA channel has a set of seven 32-bit registers (mode, status, current descriptor address, next descriptor address, source address, destination address, and byte count) to support transactions. The following sections describe the format of the DMA support registers.

12.4.8.1 DMA Mode Register (DMAMR*n*)

This section describes the DMA mode register. The mode register allows software to start the DMA transfer and to control various DMA transfer characteristics. Figure 12-10 shows the DMAMR*n* fields.

Offset: 0x100, 0x180, 0x200, 0x280 Access: User read/write

	31–28	27–24	23–21	20	19	18	17–16
R W	—	DRCNT	BWC	DMSEN	IRQS	EMSEN	DAHTS
Reset	All zeros						

	15–14	13	12	11–10	9–8	7	6–4	3	2	1	0
R W	SAHTS	DAHE	SAHE	PRC	—	EOTIE	—	TEM	CTM	CC	CS
Reset	All zeros										

Figure 12-10. DMA Mode Registers (DMAMR*n*)

Table 12-11 describes the DMAMR*n* register.

Table 12-11. DMAMR*n* Field Descriptions

Bits	Name	Description
31–28	—	Reserved
27–24	DRCNT	DMA request count. This field specifies the number of cache lines transferred per DMA request assertion if EMSEN is 1. This field is not used if EMSEN is 0. 0101 1 cache line 0110 2 cache lines 0111 4 cache lines 1000 8 cache lines 1001 16 cache lines 1010 32 cache lines Others reserved
23–21	BWC	Bandwidth control. Only applies when multiple channels are executing transfers concurrently. The field determines how many cache lines a given channel is allowed to transfer after it is granted access to the IOS interface and before it releases the interface to the next channel. This allows the user to prioritize the DMA channels. The BWC values are listed as follows: 000 1 cache line 001 2 cache lines 010 4 cache lines 011 8 cache lines 100 16 cache lines Others reserved
20	DMSEN	Direct mode snoop enable. This bit controls snooping of direct mode DMA transactions. 0 Snooping is disabled. 1 Snooping is enabled.
19	IRQS	Interrupt steer. This bit determines the destination of the DMA interrupts. 0 All DMA interrupts are routed to the on-chip interrupt controller. 1 All DMA interrupts are routed to the PCI bus through $\overline{\text{INTA}}$.

Table 12-11. DMAMR*n* Field Descriptions (continued)

Bits	Name	Description
18	EMSEN	External master start enable. This bit is cleared when the DMA transfer has completed, so it must be set again for each transfer. 0 The channel is started by software setting the CS bit. 1 The channel is started by hardware asserting the $\overline{\text{DREQ}}$ pin.
17–16	DAHTS	Destination address hold transfer size. This field indicates the transfer size used for each transaction when DAHE is 1. The byte count register must be in multiples of the size, and the destination address register must be aligned based on the size. 00 1 byte 01 2 bytes 10 4 bytes 11 8 bytes
15–14	SAHTS	Source address hold transfer size. This field indicates the transfer size used for each transaction when SAHE is 1. The byte count register must be in multiples of the size, and the source address register must be aligned based on the size. 00 1 byte 01 2 bytes 10 4 bytes 11 8 bytes
13	DAHE	Destination address hold enable. This bit allows the DMA controller to hold the destination address constant for every transfer. The size used for transfer is indicated by DAHTS. Note that hardware supports only aligned transfers for this feature. 0 Do not hold the destination address constant. 1 Hold the destination address constant. **Note:** The DMA does not support address hold for both the source and the destination at the same transfer.
12	SAHE	Source address hold enable. This bit allows the DMA controller to hold the source address constant for every transfer. The size used for transfer is indicated by SAHTS. Note that hardware supports only aligned transfers for this feature. 0 Do not hold the source address constant. 1 Hold the source address constant. **Note:** The DMA does not support address hold for both the source and the destination at the same transfer.
11–10	PRC	PCI read command. This field indicates the type of PCI read command to use. 00 1 PCI Read 01 2 PCI Read Line 10 4 PCI Read Multiple 11 8 Reserved
9–8	—	Reserved
7	EOTIE	End-of-transfer interrupt enable. This bit determines whether an interrupt is generated at the completion of a DMA transfer. End-of-transfer is defined as the end of a direct mode transfer or in chaining mode, as the end of the transfer of the last segment of a chain. 0 No EOT interrupt is generated. 1 EOT interrupt is generated.
6–4	—	Reserved
3	TEM	Transfer error mask. This bit determines the DMA response in the event of a transfer error. 0 The DMA will halt when a transfer error occurs (DMASR*n*[TE] is set). 1 The DMA will complete the transfer regardless of whether a transfer error occurs (the TE bit is not set).

Table 12-11. DMAMR*n* Field Descriptions (continued)

Bits	Name	Description
2	CTM	Channel transfer mode. 0 Chaining mode 1 Direct mode
1	CC	Channel continue. This bit applies only to chaining mode. Setting this bit indicates that the current descriptor segment should be repeated. CC is cleared by the DMA once the repeat takes effect, so it only causes a single repeat. 0 Normal chaining. 1 DMACDAR is not loaded from DMANDAR, causing a repeat of the current descriptor segment.
0	CS	Channel start. A 0-to-1 transition occurring on this bit when the channel is not busy (SR[CB] bit is 0) will start the DMA process. If the channel is busy and a 0-to-1 transition occurs, the DMA channel will restart from a previous halt condition. A 1-to-0 transition when the channel is busy (CB bit is 1) will halt the DMA process. Nothing happens if the channel is not busy and a 1-to-0 transition occurs. This bit is cleared by the DMA at the end of a transfer.

12.4.8.2 DMA Status Register (DMASR*n*)

This section describes the DMA status register. The status register reports various DMA conditions during and after the DMA transfer. Writing a 1 to a specific set bit clears the bit. Figure 12-11 shows the DMASR*n* fields.

Offset: 0x104, 0x184, 0x204, 0x284 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	All zeros															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	TE					CB	EOSI	EOCDI
W																
Reset	All zeros															

Figure 12-11. DMA Status Register (DMASR*n*)

Table 12-12 describes the DMASR*n* register.

Table 12-12. DMASR*n* Field Descriptions

Bits	Name	Description
31–8	—	Reserved
7	TE	Transfer error. Set when there is an error condition during the DMA transfer and the DMAMR*n*[TEM] bit is cleared.
6–3	—	Reserved
2	CB	Channel busy. This bit indicates whether the channel is busy. It is cleared as a result of any of the following conditions: (1) an error, (2) a halt, or (3) completion of the DMA transfer. 0 No DMA transfer is currently in progress. 1 A DMA transfer is currently in progress.

Table 12-12. DMASR*n* Field Descriptions (continued)

Bits	Name	Description
1	EOSI	End-of-segment interrupt. After transferring a segment of data, if the DMACDAR*n*[EOSIE] bit in the current descriptor address register is set, this bit is set and an interrupt is generated.
0	EOCDI	End-of-chain/direct interrupt. When the last DMA transfer is finished, either in chaining or direct mode, if DMAMR[EOTIE] is set, this bit is set and an interrupt is generated.

12.4.8.3 DMA Current Descriptor Address Register (DMACDAR*n*)

DMACDAR*n* contains the address of the current segment descriptor being transferred. In chaining mode, software must initialize this register to point to the first descriptor in the chain. After processing the first descriptor, the DMA controller moves the contents of the next descriptor address register into DMACDAR, loads the following descriptor into DMANDAR, and executes the current transfer. Figure 12-12 shows the DMACDAR*n* fields.

Offset: 0x108, 0x188, 0x208, 0x288 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	CDA															
Reset	All zeros															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	CDA											SNEN	EOSIE	0	0	0
Reset	All zeros															

Figure 12-12. DMA Current Descriptor Address Register (DMACDAR*n*)

Table 12-13 describes the DMACDAR*n* register.

Table 12-13. DMACDAR*n* Field Descriptions

Bits	Name	Description
31–5	CDA	Current descriptor address. This field contains the current descriptor address of the segment descriptor in memory. It must be aligned on an 8-word boundary.
4	SNEN	Snoop enable. 0 Snooping is disabled on DMA transactions of the current segment. 1 Snooping is enabled on DMA transactions of the current segment.
3	EOSIE	End-of-segment interrupt enable 0 No end-of-segment interrupt is generated. 1 An interrupt is generated when the current DMA transfer for the current descriptor is finished.
2–0	—	Reserved

12.4.8.4 DMA Source Address Register (DMASAR*n*)

DMASAR*n* indicates the address from which the DMA controller will be reading data. The software must ensure that this is a valid memory address. Figure 12-13 shows the DMASAR*n*.

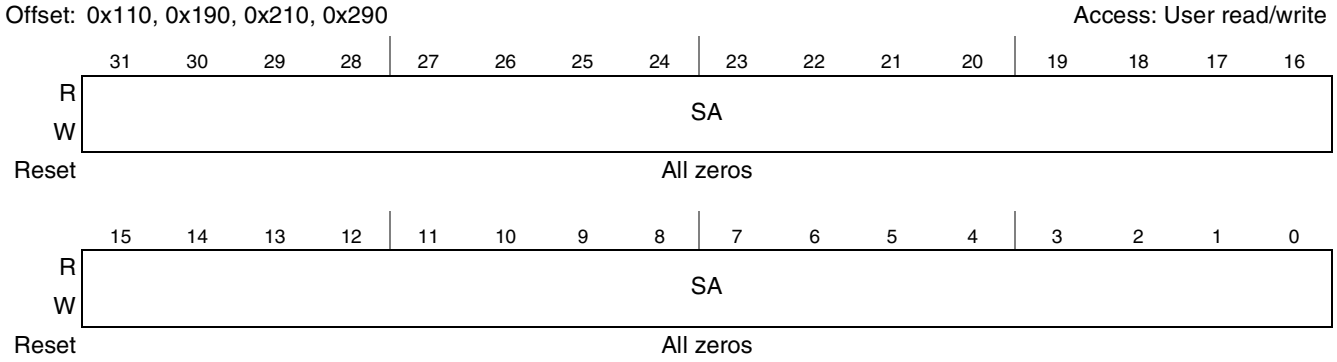


Figure 12-13. DMA Source Address Register (DMASAR*n*)

Table 12-14 describes the DMASAR*n* register.

Table 12-14. DMASAR*n* Field Descriptions

Bits	Name	Description
31–0	SA	Source address of DMA transfer. The content of this field is updated after each DMA read operation.

12.4.8.5 DMA Destination Address Register (DMADAR*n*)

DMADAR*n* indicates the address to which the DMA controller will be writing data. The software must ensure that this is a valid memory address. Figure 12-14 shows the DMADAR*n* fields.

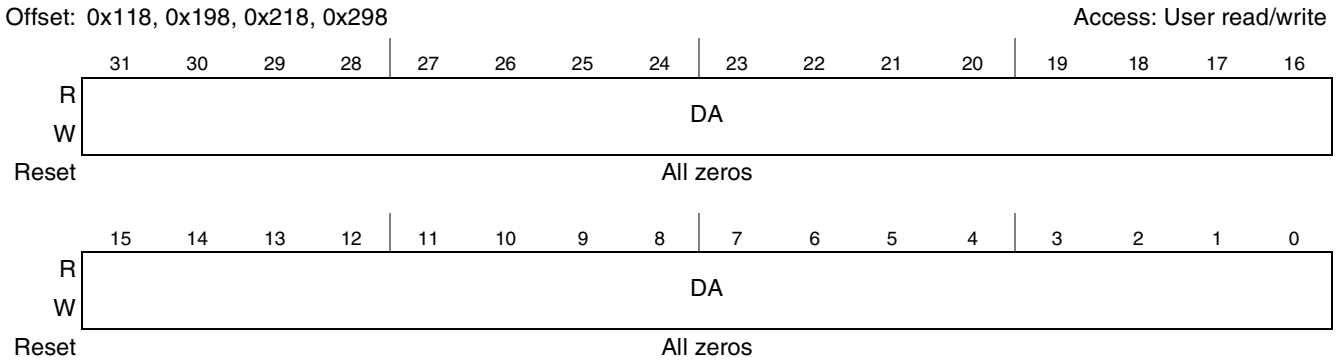


Figure 12-14. DMA Destination Address Register (DMADAR*n*)

Table 12-15 describes the DMADAR*n* register.

Table 12-15. DMASAR*n* Field Descriptions

Bits	Name	Description
31–0	DA	Destination address of DMA transfer.Updated after each DMA write operation.

12.4.8.6 DMA Byte Count Register (DMABCR*n*)

DMABCR*n* contains the number of bytes per transfer (maximum transfer size is 64 Mbytes). Figure 12-15 shows the DMABCR*n*.

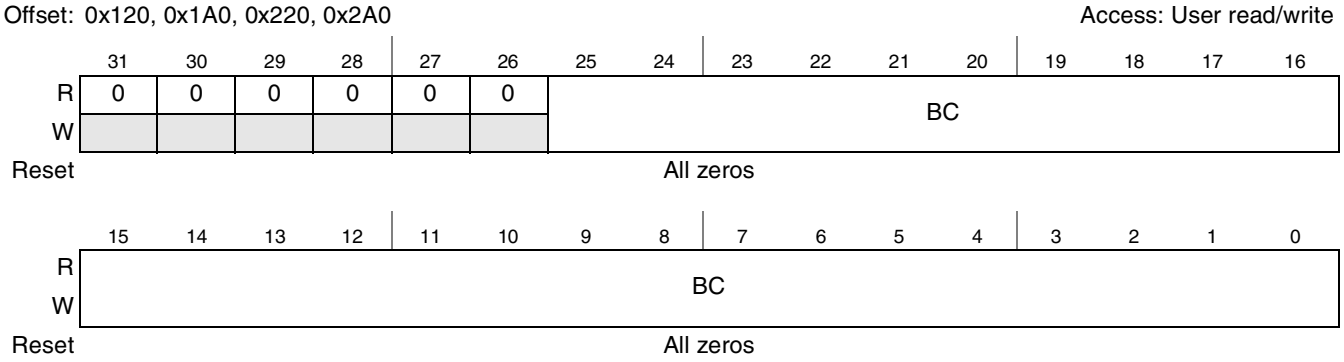


Figure 12-15. DMA Byte Count Register (DMABCR*n*)

Table 12-16 describes the DMABCR*n* register.

Table 12-16. DMABCR*n* Field Descriptions

Bits	Name	Description
31–26	—	Reserved
25–0	BC	Byte count. This field contains the number of bytes to transfer. The value in this register is decremented after each DMA read operation. Maximum transfer size is 64 Mbytes.

12.4.8.7 DMA Next Descriptor Address Register (DMANDAR*n*)

DMANDAR*n* contains the address for the next segment descriptor in the chain. In chaining mode, this register is loaded from the ‘next descriptor’ field of the descriptor to which the current descriptor address register is pointing. Figure 12-16 shows the DMANDAR*n*.

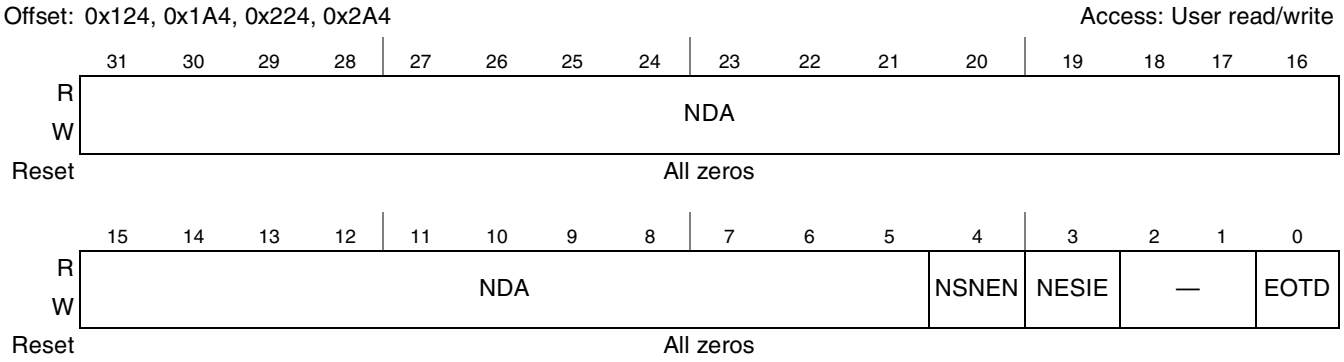


Figure 12-16. DMA Next Descriptor Address Register (DMANDAR*n*)

Table 12-17 describes the DMANDAR*n* register.

Table 12-17. DMANDAR*n* Field Descriptions

Bits	Name	Descriptions
31–5	NDA	Next descriptor address. This field contains the next descriptor address of the next segment descriptor in memory. It must be aligned on an 8-word boundary.
4	NSNEN	Next snoop enable. 0 Snooping is disabled on DMA transactions. 1 Snooping is enabled on DMA transactions.
3	NEOSIE	Next end-of-segment interrupt enable. 0 No end-of-segment interrupt is generated. 1 An interrupt is generated when the DMA transfer for the next descriptor is finished.
2–1	—	Reserved
0	EOTD	End-of-transfer descriptor. 0 This descriptor contains a link to another descriptor. 1 This descriptor is the last to be executed.

12.4.8.8 DMA General Status Register (DMAGSR)

DMAGSR provides faster access to the status bits by combining the status bits of all of the DMA channels into one register. Each byte of this register provides the value of bits 7–0 of a channel's DMA status register. These bits are cleared by writing to the individual DMA status registers. Figure 12-17 shows the DMAGSR fields.

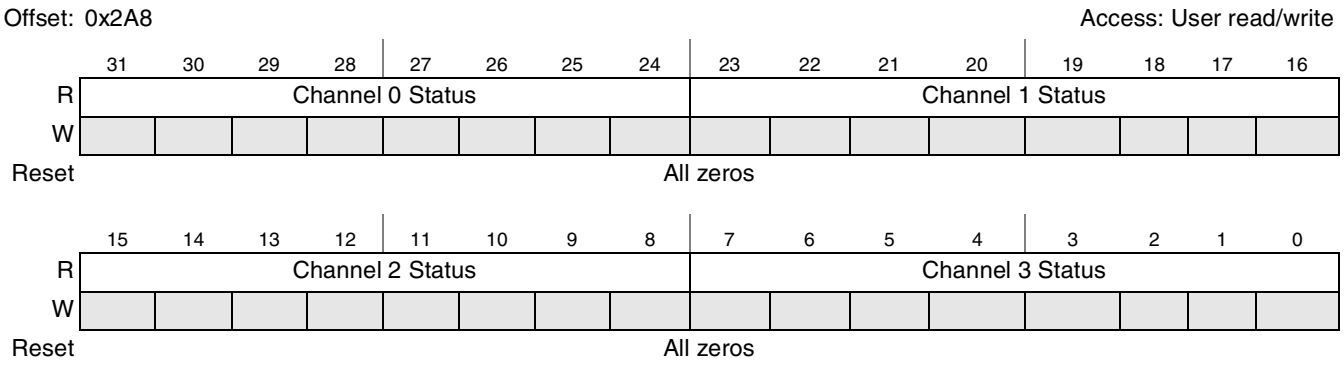


Figure 12-17. DMA General Status Register (DMAGSR)

12.5 Functional Description

12.5.1 Message Unit

An embedded processor is often part of a larger system containing many processors and distributed memory. These processors tend to work on tasks independent of the host and other peripheral processors in the system. Because of the independent nature of the tasks, it is necessary to provide a communication mechanism between the peripheral processors and the rest of the system. One such method is the use of messages. This block provides a messaging unit to further facilitate communications between host and peripheral. The message unit uses generic messages and doorbell registers.

12.5.1.1 Messaging Registers (IMR0–IMR1)

There are two 32-bit inbound message registers (IMR0–IMR1) and two 32-bit outbound message registers (OMR0–OMR1). IMR0 and IMR1allow a remote host or PCI master to write a 32-bit value that in turn causes an interrupt request to the on-chip interrupt controller that drives an interrupt line to the local processor. OMR0 and OMR1 allow the local processor to write an outbound message which, in turn, causes the outbound interrupt signal $\overline{\text{INTA}}$ to assert.

The interrupt to the local processor is cleared by writing 1 to the appropriate IMISR bit. The interrupt to PCI ($\overline{\text{INTA}}$) is cleared by writing 1 to the appropriate OMISR bit.

12.5.1.2 Doorbell Registers (IDR and ODR)

This block contains the inbound doorbell register (IDR) and the outbound doorbell register (ODR). The inbound doorbell allows a remote processor to set a bit in the register from the PCI bus. This, in turn generates an interrupt request to the on-chip interrupt controller that drives an interrupt line to the local processor. The local processor can write to the ODR, which causes the outbound interrupt signal $\overline{\text{INTA}}$ to assert, thus interrupting the remote processor on the PCI bus.

The interrupt to the local processor is cleared by writing 1 to the appropriate IDR bit. The interrupt to PCI ($\overline{\text{INTA}}$) is cleared by writing 1 to the appropriate ODR bit.

12.5.2 DMA Controller

The DMA controller transfers blocks of data independent of the local processor or PCI hosts. Data movement occurs on the PCI bus and/or CSB. The DMA module has four high-speed DMA channels, which share buffer space in the IOS to facilitate the gathering and sending of data. Both the local processor and PCI masters can initiate a DMA transfer.

Features of the DMA controller include the following:

- Four channels
- Concurrent execution across multiple channels with programmable bandwidth control
- All channels are accessible by local processor and remote PCI masters
- Unaligned transfer capability
- Data chaining and direct mode
- Interrupt on completed segment, chain, and error

Figure 12-18 shows a diagram of the DMA controller in the integrated device.

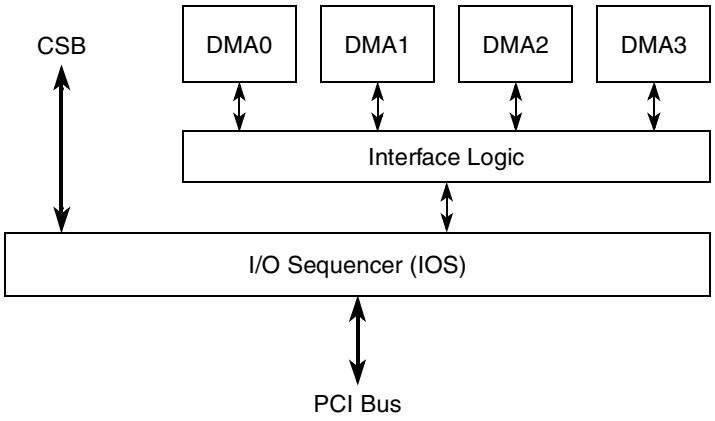


Figure 12-18. DMA Controller Block Diagram

12.5.3 DMA Operation

The DMA controller operates in the following two modes:

- Direct mode, in direct mode, the DMA controller does not read a chain of descriptors from memory but instead uses the current parameters in the DMA registers to start a DMA transfer. The DMA transfer finishes after all the bytes specified in the byte count register have been transferred. See Section 12.6.1, “Initialization Steps in Direct Mode,” for more details on initialization steps.
- Chaining mode, in chaining mode, the DMA controller loads descriptors from memory prior to a DMA transfer. The DMA controller begins the transfer according to the descriptor information loaded for each segment. Once the current segment is finished, the DMA controller reads the next descriptor from memory and begins another DMA transfer. The process is finished if the current descriptor is the last one in the chain. See Section 12.6.2, “Initialization Steps in Chaining Mode,” for more details on initialization steps.

In both modes, setting the start bit in the DMA mode register begins the DMA transfer.

The DMA controller supports unaligned transfers for both the source and destination addresses. It gathers data beginning at the source address and aligns the data accordingly before sending it to the destination address. The DMA controller assumes that the source and destination addresses are valid PCI or CSB memory addresses.

Accesses to CSB memory depend on the alignment of the source and destination addresses and the size of the transfer. The DMA controller transfers a full cache line whenever possible. Misaligned destination addresses result in sub-transfers of less than a cache line on the initial and final beats of the transfer; intermediate beats transfer full cache lines. Configuring a DMA channel for address hold mode DMAMR*n* precludes cache line transfers.

PCI memory read operations depend on the PRC (PCI read command) field in the mode register, the alignment of the source address, and the size of the transfer. The DMA controller attempts to read a full cache line whenever possible. Writing to PCI memory depends on the alignment of the destination address and the size of the transfer.

12.5.3.1 External Control

The DMA transfer of any channel, in either direct mode or chaining mode, can be controlled by the DMA request input signals. External control is enabled by setting the external master start enable (EMSEN) bit instead of the channel start (CS) bit in the DMA mode register (DMAMR*n*).

When using external control, the following restrictions apply:

- Both the source and destination addresses must be aligned to 32-byte boundaries.
- In chaining mode all byte count values except the last must be multiples of 32 bytes.

A falling edge on $\overline{\text{DREQ}n}$ sets DMAMR*n*[CS] to start the transfer and asserts the corresponding $\overline{\text{DACK}n}$ output signal. The number of cache lines specified by the DMA request count (DRCNT) bit in the DMA mode register (DMAMR*n*)—or the remaining byte count, if smaller— is transferred, and the CS bit and the $\overline{\text{DACK}n}$ output signal are then cleared and negated to halt the transfer until the next $\overline{\text{DREQ}n}$ assertion.

When using the $\overline{\text{DACK}n}$ handshake signal, $\overline{\text{DREQ}n}$ should remain asserted until $\overline{\text{DACK}n}$ is asserted, at which point $\overline{\text{DREQ}n}$ may be negated. $\overline{\text{DREQ}n}$ may be asserted again to resume the transfer or start a new transfer once $\overline{\text{DACK}n}$ has been negated.

If the $\overline{\text{DACK}n}$ handshake signal is not used, $\overline{\text{DREQ}n}$ should remain asserted until the first transaction of the DMA transfer appears on the external interface, at which point $\overline{\text{DREQ}n}$ may be negated. $\overline{\text{DREQ}n}$ may be asserted again to resume the transfer or start a new transfer as early as one clock cycle after its negation, even if the current transfer is still active. The DMA controller is able to record a new assertion of $\overline{\text{DREQ}n}$ while a transfer is in progress, with the effect of setting DMAMR*n*[CS] again once the transfer has been halted.

Once a transfer has been completed, DMAMR*n*[EMSEN] is cleared by the DMA controller, and DMAMR*n*[CS] will not be set again until DMAMR*n*[EMSEN] has been set by the user. The assertion of $\overline{\text{DREQ}n}$ and the setting of DMAMR*n*[EMSEN] may occur in either order; whichever occurs later will trigger the DMA transfer.

The $\overline{\text{DDONE}n}$ output signal is asserted when the DMA transfer has completed, that is, all bytes specified in the byte count register or the descriptors have been transferred. This signal could be used as an indication that the number of cache lines transferred might be smaller than that specified by the DRCNT field.

NOTE

The $\overline{\text{DACK}n}$ and $\overline{\text{DDONE}n}$ output signals are intended as handshake signals for $\overline{\text{DREQ}n}$. They are asserted and negated according to the DMA controller internal logic. These signals are not synchronized to the transactions appearing on the external pins of the device. Specifically, the negation of $\overline{\text{DACK}n}$ or the assertion of $\overline{\text{DDONE}n}$ does not mean that all transactions of the DMA transfer have been completed as seen on the external pins.

See Section 12.6.3, “Initialization Steps in Direct Mode with External Control,” and Section 12.6.4, “Initialization Steps in Chaining Mode with External Control,” for more details on initialization steps.

12.5.3.2 DMA Coherency

The four DMA channels use up to four cache lines (128 bytes) of buffer space in the IOS in addition to 16 bytes of local buffer space. Because no address snooping occurs in these internal queues, data posted in these queues is not visible to the rest of the system while a DMA transfer is in progress. It is the responsibility of application software to ensure the coherency of the region being transferred during the DMA process.

Snooping of the CPU or processor data cache is selectable during DMA transactions. A snoop bit is provided in the DMA current descriptor address register (DMACDAR*n*) and the DMA next descriptor address register (DMANDAR*n*) that allows software to control when the cache is snooped on a per segment basis.

12.5.3.3 Halt and Error Conditions

DMA transfers are halted either by clearing the CS (channel start) bit in the DMA mode register (DMAMR*n*) or when encountering an error condition. In either case, the application software can do one of the following:

- Continue the DMA transfer
- Reconfigure the DMA for a new transfer
- Leave the channel in the halted state

When a DMA channel is halted, its programming model is completely accessible. If the DMA is halted due to an error condition, the TE (transfer error) bit in the DMA status register (DMASR*n*) must be cleared before the transfer can be resumed or a new transfer initiated. Note that the TE bit is not cleared automatically by hardware.

12.5.4 DMA Segment Descriptors

DMA segment descriptors contain the source and destination addresses of the data segment, the segment byte count, and a link to the next descriptor. Segment descriptors are built on cache-line (32-byte) boundaries in either CSB or PCI memory and are linked together into chains using the next-descriptor-address field.

Table 12-18. DMA Segment Descriptor Fields

Descriptor Field	Description
Source address	Contains the source address of the DMA transfer. After the DMA controller reads the descriptor from memory, this field will be loaded into the DMA Source Address Register (DMASARn).
Destination address	Contains the destination address of the DMA transfer. After the DMA controller reads the descriptor from memory, this field will be loaded into the DMA Destination Address Register (DMADARn).
Next descriptor address	Points to the next descriptor in memory. After the DMA controller reads the descriptor from memory, this field will be loaded into the DMA Next Descriptor Address Register (DMANDARn).
Byte count	Contains the number of bytes to transfer. After the DMA controller reads the descriptor from memory, this field will be loaded into the DMA Byte Count Register (DMABCRn).

Application software initializes the current DMA current descriptor address register (DMACDAR*n*) to point to the first descriptor in the chain. For each descriptor in the chain, the DMA controller starts a new DMA transfer with the control parameters specified by the descriptor. The DMA controller traverses the descriptor chain until reaching the last descriptor (with its EOTD bit set).

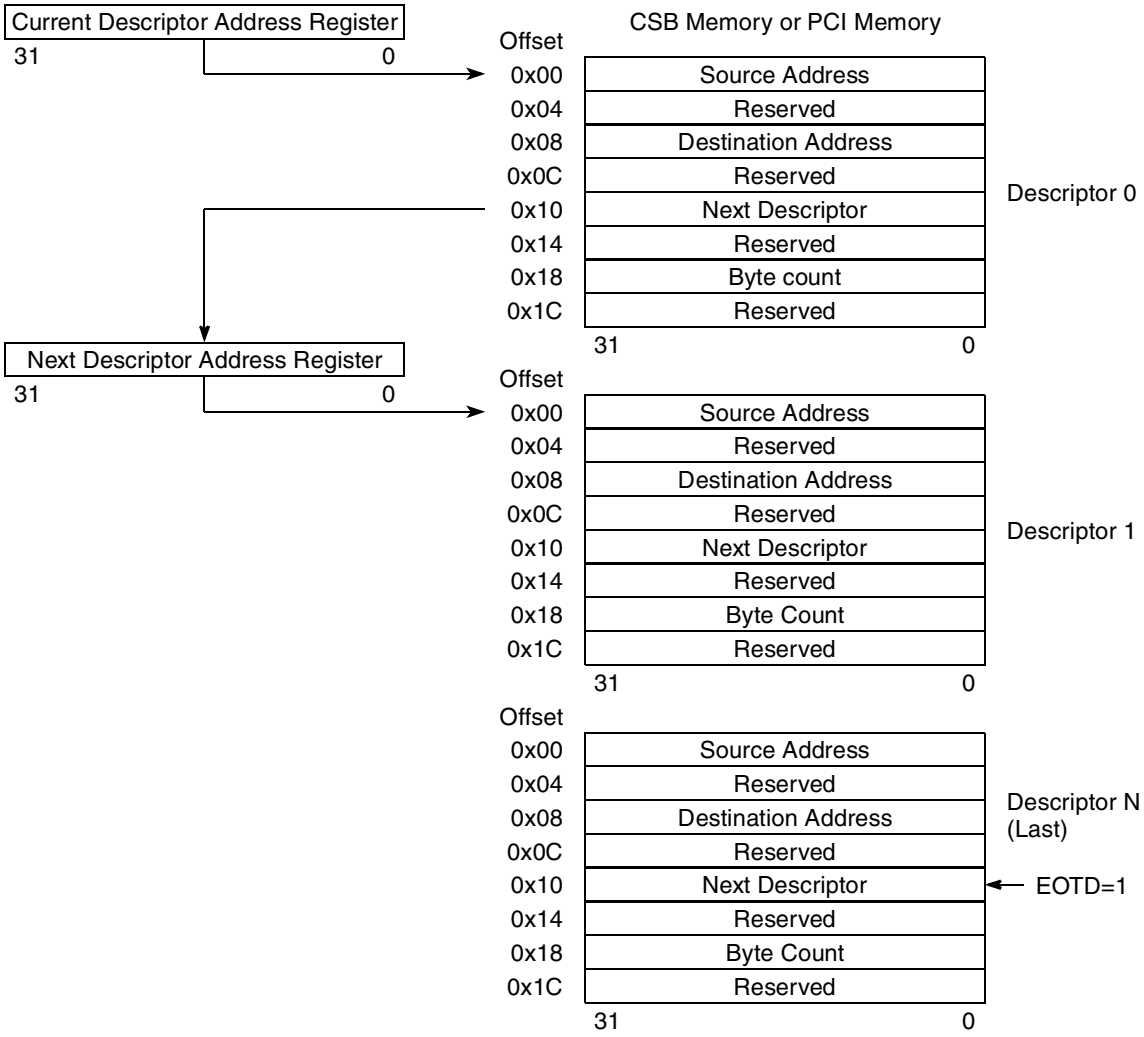


Figure 12-19. DMA Chain of Segment Descriptors

12.5.4.1 Descriptor in Big-Endian Mode

In big-endian mode, the descriptor in CSB memory should be programmed such that data appears in ascending significant-byte order. If segment descriptors are written to memory located in the CSB, they should be treated like they are translated from big-endian to little-endian mode.

Example: Big-endian mode descriptor's data structure. Note that the descriptor structure must be aligned on an 8-word boundary.

```
struct {
          double a;           /* 0x1122334455667788 double word*/
          double b;           /* 0x55667788aabbccdd double word*/
          double c;           /* 0x8765432101234567 double word */
```

```
          double d;           /* 0x0123456789abcdef double word */
} Descriptor;
Results:  Source Address = 0x44332211 <MSB..LSB>
          Destination Address = 0x88776655 <MSB..LSB>
          Next Descriptor Address = 0x21436587 <MSB..LSB>
          Byte Count = 0x67452301 <MSB..LSB>
```

12.5.4.2 Descriptor in Little-Endian Mode

In little-endian mode, each segment descriptor should be programmed in descending significant-byte order.

Example: Little-endian mode descriptor's data structure. Note that the descriptor structure must be aligned on an 8-word boundary.

```
struct {
          double a;           /* 0x8877665544332211 double word*/
          double b;           /* 0x1122334488776655 double word*/
          double c;           /* 0x7654321012345678 double word */
          double d;           /* 0x0123456776543210 double word */
} Descriptor;
Results:  Source Address = 0x44332211 <MSB..LSB>
          Destination Address = 0x88776655 <MSB..LSB>
          Next Descriptor Address = 0x12345678 <MSB..LSB>
          Byte Count = 0x76543210 <MSB..LSB>
```

12.6 Initialization/Application Information

12.6.1 Initialization Steps in Direct Mode

The initialization steps of a DMA transfer in direct mode are described as follows:

1. Poll the CB (channel busy) bit in the DMA status register (DMASR*n*) to make sure the DMA channel is idle
2. Initialize the DMASAR*n*, DMADAR*n*, and the DMABCR*n*
3. Initialize DMAMR*n*[CTM]) to indicate direct mode. Other control parameters in the mode register can also be initialized here if necessary
4. First clear then set the DMAMR*n*[CS] to start the DMA transfer

12.6.2 Initialization Steps in Chaining Mode

The initialization steps of a DMA transfer in chaining mode are described as follows:

1. Build a chain of descriptor segments in memory. Refer to Section 12.5.4, "DMA Segment Descriptors."
2. Poll the DMASR*n*[CB] to make sure the DMA channel is idle
3. Initialize the DMACDAR*n* to point to the first descriptor in the chain
4. Initialize the DMAMR*n*[CTM] to indicate chaining mode. Other control parameters in the mode register can also be initialized here if necessary
5. First clear then set the DMAMR*n*[CS] to start the DMA transfer

12.6.3 Initialization Steps in Direct Mode with External Control

The initialization steps of a DMA transfer in direct mode with external control are described as follows:

1. Poll the CB (channel busy) bit in the DMA status register (DMASR*n*) to make sure the DMA channel is idle.
2. Initialize the DMA source address register (DMASAR*n*), the DMA destination address register (DMADAR*n*), and the DMA byte count register (DMABCR*n*).
3. Set DMAMR*n*[CTM]) to indicate direct mode, program the DRCNT field, and set the EMSEN bit. Other control parameters in the mode register can also be initialized here if necessary.

12.6.4 Initialization Steps in Chaining Mode with External Control

The initialization steps of a DMA transfer in chaining mode with external control are described as follows:

1. Build a chain of descriptor segments in memory. Refer to Section 12.5.4, “DMA Segment Descriptors,” for more information.
2. Poll the CB (channel busy) bit in the DMA status register (DMASR*n*) to make sure the DMA channel is idle.
3. Initialize the DMA current descriptor address register (DMACDAR*n*) to point to the first descriptor in the chain.
4. Clear the DMAMR*n*[CTM] to indicate chaining mode, program the DRCNT field, and set the EMSEN bit. Other control parameters in the mode register can also be initialized here if necessary.

Chapter 13
PCI Bus Interface

The PCI interface is compatible with the *PCI Local Bus Specification*, Rev. 2.3. It is beyond the scope of this manual to document the intricacies of PCI. This chapter describes the PCI controller and provides a basic description of the PCI bus operations. The specific emphasis is directed at how this device implements the PCI specification. Designers of systems incorporating PCI devices should refer to the respective specifications for a thorough description of the PCI buses.

NOTE

Much of the available PCI literature refers to a 16-bit quantity as a WORD and a 32-bit quantity as a DWORD. Because this is inconsistent with the terminology in this manual, the terms ‘word’ and ‘double word’ are not used in this chapter. Instead, the number of bits or bytes indicates the exact quantity.

13.1 Introduction

The PCI controller acts as a bridge between the PCI interface and the CSB. The I/O sequencer buffers the data. Figure 13-1 is a high-level block diagram of the PCI controller.

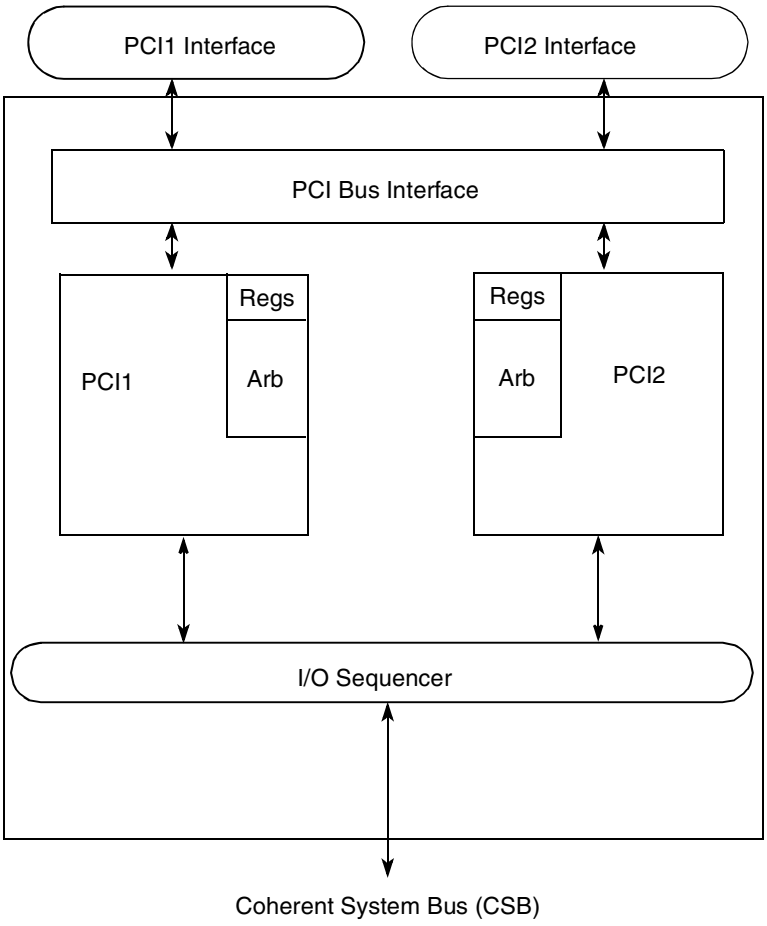


Figure 13-1. PCI Controller Block Diagram

The MPC8349EA contains two PCI interfaces. If both are used, they are both 32-bit ports. If only PCI1 is used, it may be configured to as a 32-bit or 64-bit port. This chapter refers to both 32-bit and 64-bit ports; some of the signals are not used when the PCI controller is operating as a 32-bit port.

Note that PCI1 supports up to five external masters, while PCI2 supports up to three external masters. The unused request signals of PCI2 are internally tied to an inactive state and do not affect the operation of the arbiter.

The PCI controller connects the processor and memory system to the I/O components through the PCI system bus. This interface acts as both initiator (master) and target (slave) device. The PCI controller uses a 32- or 64-bit multiplexed, address/data bus that can run at frequencies up to 66-MHz. The interface provides address and data parity with error checking and reporting. The interface provides for three physical address spaces—64-bit address memory, 32-bit address I/O, and PCI configuration space.

Note that the device supports a 64-bit address/data bus.

The PCI1 interface can function as either a PCI host bridge referred to as host mode or a peripheral device on the PCI bus referred to as agent mode. See Section 13.4.4.4, “Host Mode Configuration Access,” for more information. The PCI2 interface only supports host mode. Note that the PCI controller can be configured from the PCI bus while in agent mode. An address translation mechanism is provided to map PCI memory windows between the PCI bus and the internal bus.

13.1.1 Features

The PCI controller includes the following features:

- PCI specification revision 2.3 compliant
- 64- and 32-bit PCI interface support
- Host and agent mode support on PCI1; host mode support only on PCI2
- Supports accesses to all PCI address spaces
- 64-bit dual-address cycle (DAC) support (as a target only)
- Internal configuration registers accessible from PCI
- On-chip arbitration supporting five masters on PCI1, three masters on PCI2
- Arbiter supports two-level priority request/grant signal pairs
- Supports PCI-to-memory and memory-to-PCI streaming
- Memory prefetching of PCI read accesses and support for delayed read transactions
- Supports posting of processor-to-PCI and PCI-to-memory writes
- Supports selectable snooping for inbound transactions
- Address translation units for address mapping between host and peripheral
- Supports parity
- PCI 3.3-V compatible

13.1.2 Modes of Operation

The PCI controller modes of operation are determined at reset by the reset configuration word high (RCWH) as described in Section 4.3.2, “Reset Configuration Words.” Table 13-1 summarizes these modes.

Table 13-1. PCI Controller Modes

Parameter	Description	Section/Page
Host/agent configuration	Selects between host and agent mode for the PCI1 interface.	4.3.2.2.1/4-16
PCI Interface selection	Selects between one 64-bit PCI interface or two 32-bit PCI interfaces.	4.3.2.2/4-14
PCI arbiter enable	Enables the on-chip PCI bus arbiter	4.3.2.2/4-14

13.1.2.1 Host/Agent Mode Configuration

The PCI controller can function as either a PCI host bridge (referred to as host mode) or a peripheral device on the PCI bus (referred to as agent mode). Note that host/agent mode selection is determined at power-up as summarized in Section 4.3.2.2.1, "PCI Host/Agent Configuration."

When the device powers up in host mode, all inbound configuration accesses are ignored (and thus master aborted). When the device powers up in agent mode, it acknowledges inbound configuration accesses. Note that in PCI agent mode, the PCI controller ignores all PCI memory accesses except those to the memory-mapped registers until inbound address translation is enabled. In agent mode, configuration cycles are acknowledged if CFG_LOCK is 0 (see Section 13.3.3.24, "PCI Function Configuration Register"), either from reset configuration or after being cleared by software.

Note that only the primary PCI interface (PCI1) supports agent mode.

13.1.2.2 Single 64-Bit/Dual 32-Bit Interface Configuration

The interface can be configured to be a single 64-bit PCI interface or two independent 32-bit PCI interfaces. See Table 4-25 for pin assignments for the PCI-64 configuration. The initial value for PCI Interface selection is determined by the value in RCWH[PCI64]. See Section 1.3.2.8, "PCI 64-Bit Configuration," for more information.

13.1.2.3 PCI Arbiter Configuration

The interface can be configured to use an on-chip or off-chip PCI arbiter. Arbitration for PCI1 is determined by the value in RCWH[PCI1ARB]; arbitration for PCI2 is determined by the value in RCWH[PCI2ARB]. See Section 4.3.2.2, "Reset Configuration Word High Register (RCWHR)," for more information.

13.2 External Signal Description

Table 13-2 shows the properties of the PCI signals.

Table 13-2. Signal Properties

Name	Function	Reset State	Pull Up
CPCI_HS_ENUM	CompactPCI hot swap enumerator	High impedance	Required
CPCI_HS_ES	CompactPCI hot swap ejector switch	—	—
CPCI_HS_LED	CompactPCI hot swap LED	Asserted	—
M66EN	66-MHz enable	—	—
$\overline{\text{PCI1_REQ64}}$	PCI1 request 64-bit transfer	High impedance[1]	Required
$\overline{\text{PCI1_ACK64}}$	PCI1 acknowledge 64-bit transfer	High impedance	Required
PCI1_AD[63:0]	PCI1 address / data	High impedance	—
PCI2_AD[31:0]	PCI2 address / data	High impedance	—
PCI$\overline{n}$_C/$\overline{\text{BE}}$[3:0]	PCI bus command / byte enable	High impedance	—
PCI1_CBE[7:0]	PCI1 bus command / byte enable	High impedance	—

Table 13-2. Signal Properties

Name	Function	Reset State	Pull Up
PCI*n*_DEVSEL	PCI*n* device select	High impedance	Required
PCI*n*_FRAME	PCI*n* cycle frame	High impedance	Required
PCI1_REQ[0:4]	PCI1 arbiter requests	Configuration-dependent	Required on inputs
PCI2_REQ[0:2]	PCI2 arbiter requests	Configuration-dependent	Required on inputs
PCI1_GNT[0:4]	PCI1 arbiter grants	Configuration-dependent	—
PCI2_GNT[0:2]	PCI2 arbiter grants	Configuration-dependent	—
PCI1_IDSEL	PCI1 initialization device select	—	—
PCI1_INTA	PCI1 interrupt A	High impedance	Required
PCI*n*_IRDY	PCI*n* initiator ready	High impedance	Required
PCI*n*_PAR	PCI*n* parity	High impedance	—
PCI1_PAR64	PCI1 parity upper DWORD	High impedance	—
PCI*n*_PERR	PCI*n* parity error	High impedance	Required
PCI*n*_RESET_OUT	PCI*n* reset output	Asserted	
PCI*n*_SERR	PCI*n* system error	High impedance	Required
PCI*n*_STOP	PCI*n* stop	High impedance	Required
PCI*n*_TRDY	PCI*n* target ready	High impedance	Required

[1] Note that in PCI 64-bit host mode, PCI_REQ64 is asserted (low) immediately after HRESET negation and remains asserted until one PCI clock cycle after the negation of PCI1_RESET_OUT

Figure 13-2 shows the external PCI signals.

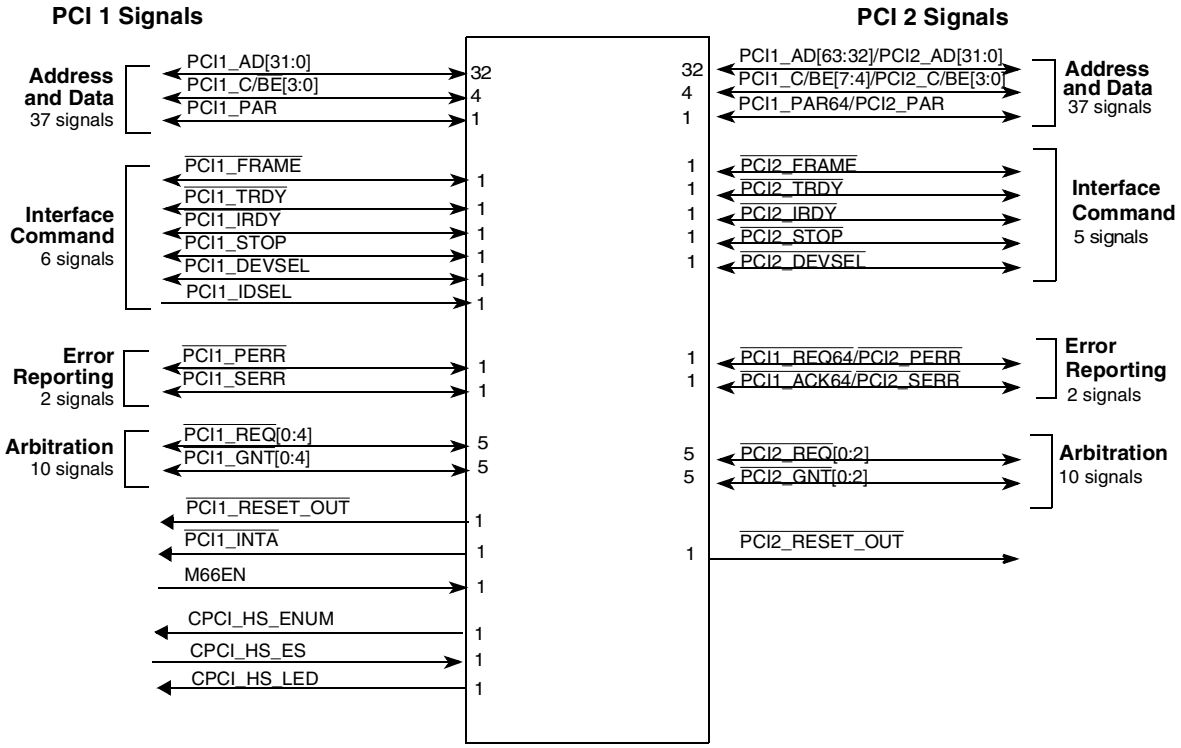


Figure 13-2. PCI Interface External Signals

Table 13-3 contains detailed descriptions of the external PCI interface signals.

Table 13-3. PCI1 and PCI2 Interface Signals—Detailed Signal Descriptions

Signal	I/O	Description	
CPCI_HS_ENUM	O	CompactPCI hot swap enumerator. Used for the hot swap interface to connect to the host as the enumeration request in a compact PCI system. This signal is used for agent mode only.	
		State Meaning	Asserted—This card was inserted and needs to be configured, or this card is about to be extracted and needs to be removed from the system resources list. Negated—No action is needed.
		Timing	Assertion/Negation—No timing is specified
CPCI_HS_ES	I	CompactPCI hot swap ejector switch. Used for agent mode only. In a compact PCI system this input signal is used for the hot swap interface to connect to the ejector switch logic.	
		State Meaning	Asserted—The switch is open. Negated—The switch is closed.
		Timing	Assertion/Negation—No timing is specified

Table 13-3. PCI1 and PCI2 Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
CPCI_HS_LED	O	CompactPCI hot swap LED. Used for the hot swap interface to connect to the hot swap LED in a CompactPCI system. This signal is used for agent mode only.	
		State Meaning	Asserted—Output is driving logic 1 to illuminate the hot swap LED. Negated—Output is driving logic 0 to turn off the hot swap LED.
		Timing	Assertion/Negation—No timing is specified
M66EN	I	66-MHz enable. Determines the AC timing of the PCI interface.	
		State Meaning	When PCI1 and PCI2 have the same frequency (OCCR[PCI1CR] = OCCR[PCI2CR]): Asserted—The PCI1 interface signals use the 66-MHz PCI AC timing parameters. The PCI2 interface signals use the 66-MHz PCI AC timing parameters. Negated—The PCI1 interface signals use the 33-MHz PCI AC timing parameters. The PCI2 interface signals use the 33-MHz PCI AC timing parameters. When PCI1 and PCI2 don't have the same frequency (OCCR[PCI1CR] = 0 and OCCR[PCI2CR] = 1 or OCCR[PCI1CR] = 1 and OCCR[PCI2CR] = 0): Asserted—The PCI1 interface signals use the 66-MHz PCI AC timing parameters. The PCI2 interface signals use the 33-MHz PCI AC timing parameters. Negated—The PCI1 interface signals use the 33-MHz PCI AC timing parameters. The PCI2 interface signals use the 66-MHz PCI AC timing parameters.
		Timing	Assertion/Negation—Constant
PCI*n*_AD[63:0]	I/O	PCI address/data bus. During an address phase, these signals contain a physical address. During a data phase, these signals contain the data bytes. If PCI_64 is 0, PCI_AD[63:32] are not used.	
	O	Outputs for the bi-directional PCI address/data bus.	
		State Meaning	Asserted/Negated—Represents the physical address during the address phase of a PCI transaction. During the data phase(s) of a PCI transaction, the PCI address/data bus contain the data being written. PCI_AD[7:0] define the LSB andPCI*n*_AD[63:56] define the MSB.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I	Inputs for the bi-directional PCI address/data bus.	
		State Meaning	Asserted/Negated—Represents the address to be decoded as a check for device select during the address phase of a PCI transaction or the data being received during the data phase(s) of a PCI transaction. PCI_AD[7:0] define the LSB andPCI*n*_AD[63:56] define the MSB.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3

Table 13-3. PCI1 and PCI2 Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
PCI*n*_C/$\overline{\text{BE}}$[7:0]	I/O	PCI bus command/byte enable.	
	O	Outputs for the bi-directional command/byte enable.	
		State Meaning	Asserted/Negated—During the address phase, PCI_CBE[7:0] define the bus command. Byte enables determine which byte lanes carry meaningful data for PCI bus data phases. The PCI_CBE[0] signal applies to the LSB. If PCI_64 is 0, PCI_CBE[7:4] are not used.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I	Inputs for the bi-directional command/byte enable.	
		State Meaning	Asserted/Negated—During the address phase, PCI_CBE[7:0] indicate the command that another master is sending. During the PCI bus data phase, PCI_CBE[7:0] indicate which byte lanes are valid. If PCI_64 is 0, PCI_CBE[7:4] are not used.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
$\overline{\text{PCI}n\text{_DEVSEL}}$	I/O	PCI*n* device select.	
	O	Outputs for the bi-directional device select.	
		State Meaning	Asserted—The PCI*n* controller has decoded the address and is the target of the current access. Negated—The PCI*n* controller has decoded the address and is not the target of the current access.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I	Inputs for the bi-directional device select.	
		State Meaning	Asserted—Some PCI agents (other than this PCI*n* controller) have decoded its address as the target of the current access. Negated—No PCI agent has been selected.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
$\overline{\text{PCI}n\text{_FRAME}}$	I/O	PCI*n* cycle frame. Used by the current PCI master to indicate the beginning and duration of an access.	
	O	Outputs for the bi-directional frame.	
		State Meaning	Asserted—The PCI*n* controller acting as a PCI master which is initiating a bus transaction. While $\overline{\text{PCI_FRAME}}$ is asserted, data transfers may continue. Negated—If $\overline{\text{PCI_IRDY}}$ is asserted, indicates that the PCI transaction is in the final data phase; if $\overline{\text{PCI_IRDY}}$ is negated, indicates that the PCI bus is idle.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I	Inputs for the bi-directional frame.	
		State Meaning	Asserted—Another PCI master is initiating a bus transaction. Negated—The transaction is in the final data phase or that the bus is idle.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3

Table 13-3. PCI1 and PCI2 Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
PCI*n*_GNT0	I/O	PCI*n* arbiter grants. Output signal on this PCI*n* controller when the arbiter is enabled. Input signal when the arbiter is disabled. **Note:** PCI_GNT[0] is a point-to-point signal. Every master has its own bus grant signal.	
	O	Outputs for the bi-directional arbiter grants.	
		State Meaning	Asserted—The PCI*n* controller granted control of the PCI bus to agent 0. Negated—The PCI*n* controller did not grant control of the PCI bus to agent 0.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I	Inputs for the bi-directional arbiter grants.	
		State Meaning	Asserted—The PCI*n* controller has been granted control of the PCI bus by an external arbiter. Negated—The PCI*n* controller has not been granted control of the PCI bus by an external arbiter.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
PCI*n*_GNT[1:4]	O	PCI*n* arbiter grants. Output signals on this PCI*n* controller when the arbiter is enabled. Note that PCI_GNT*n* is a point-to-point signal. Every master has its own bus grant signal.	
		State Meaning	Asserted—The PCI*n* controller granted control of the PCI bus to agent *n*. Negated—The PCI*n* controller did not grant control of the PCI bus to agent *n*.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
PCI1_IDSEL	I	PCI1 initialization device select. Used as a chip select during a PCI configuration cycle in agent mode. This signal should be tied low in host mode.	
		State Meaning	Asserted—The PCI1 controller is being selected as a target of a configuration read or write transactions. Negated—The PCI1 controller is not being selected as a target of configuration read or write transactions.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
PCI1_INTA	O	PCI1 interrupt A.	
		State Meaning	Asserted—The PCI1 controller signals an interrupt to the PCI host. Negated—The PCI1 controller is not currently signalling an interrupt.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3

Table 13-3. PCI1 and PCI2 Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
PCI*n*_IRDY	I/O	PCI initiator ready. This signal is driven by the PCI controller when it is the initiator of a PCI transfer.	
	O	Outputs for the bi-directional initiator ready.	
		State Meaning	Asserted—The PCI*n* controller, acting as a PCI master, can complete the current data phase of a PCI transaction. During a write, this PCI*n* controller asserts PCI_IRDY to indicate that valid data is present on PCI1_AD[63:0]. During a read, this PCI*n* controller asserts PCI_IRDY to indicate that it is prepared to accept data. Negated—The PCI target needs to wait before this PCI*n* controller, acting as a PCI master, can complete the current data phase. During a write, this PCI*n* controller negates PCI_IRDY to insert a wait cycle when it cannot provide valid data to the target. During a read, this PCI*n* controller negates PCI_IRDY to insert a wait cycle when it cannot accept data from the target.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I	Inputs for the bi-directional initiator ready.	
		State Meaning	Asserted—Another PCI master can complete the current data phase of a transaction. Negated—If PCI_FRAME is asserted, indicates a wait cycle from another master. If PCI_FRAME is negated, indicates that the PCI bus is idle.
PCI*n*_PAR	I/O	PCI*n* parity.	
	O	Outputs for the bi-directional parity.	
		State Meaning	Asserted—Odd parity across PCI_AD[31:0] and PCI_CBE[3:0] during address and data phases. Negated—Even parity across PCI_AD[31:0] and PCI_AD[31:0] during address and data phases.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I	Inputs for the bi-directional parity.	
		State Meaning	Asserted—Odd parity driven by another PCI master or the PCI target during address and data phases. Negated—Even parity driven by another PCI master or the PCI target during address and data phases.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3

Table 13-3. PCI1 and PCI2 Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
PCI1_PAR64	I/O	Upper DWORD parity. The even parity bit that protects the upper 32 bits of data and upper 4 bits of command/byte enable.	
	O	Outputs for the bi-directional upper DWORD parity.	
		State Meaning	Asserted—Odd parity across the PCI_AD[63:32] and PCI_C/$\overline{\text{BE}}$[7:4] signals during address and data phases. Negated—Even parity across the PCI_AD[63:32] and PCI_C/$\overline{\text{BE}}$[7:4] signals during address and data phases.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I	Inputs for the bi-directional upper DWORD parity.	
		State Meaning	Asserted—Odd parity driven by another PCI master or the PCI target during read data phases. Negated—Even parity driven by another PCI master or the PCI target during read data phases.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
$\overline{\text{PCI}n\text{_PERR}}$	I/O	PCI*n* parity error	
	O	Outputs for the bi-directional parity error.	
		State Meaning	Asserted—The PCI*n* controller, acting as a PCI agent, detected a data parity error. (driven by the PCI initiator on reads; driven by the PCI target on writes.) Negated—No error.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I	Inputs for the bi-directional parity error.	
		State Meaning	Asserted—Another PCI agent detects a data parity error while this PCI*n* controller is sourcing data (this PCI*n* controller was acting as the PCI initiator during a write, or is acting as the PCI target during a read). Negated—No error.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
$\overline{\text{PCI}n\text{_REQ}}$0	I/O	PCI*n* bus request**.** Input signal on this PCI*n* controller when the arbiter is enabled. Output signal when the arbiter is disabled. Note that $\overline{\text{PCI_REQ}n}$ is a point-to-point signal. Every master has its own bus request signal.	
	O	Outputs for the bi-directional bus request.	
		State Meaning	Asserted—The PCI controller is requesting control of the PCI bus to perform a transaction. Negated—The PCI controller does not require use of the PCI bus.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I	Input for the bi-directional bus request.	
		State Meaning	Asserted—Agent 0 is requesting control of the PCI bus to perform a transaction. Negated—Agent 0 does not require use of the PCI bus.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3

Table 13-3. PCI1 and PCI2 Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
PCI*n*_REQ[1:4]	I	PCI*n* bus request. Input signals on this PCI*n* controller when the arbiter is enabled. Note that PCI_REQ[n] is a point-to-point signal. Every master has its own bus request signal. Following is the state meaning for the PCI_REQ[n] input.	
		State Meaning	Asserted—An agent *n* is requesting control of the PCI bus to perform a transaction. Negated—An agent *n* does not require use of the PCI bus.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
PCI1_REQ64	I/O	64-bit transaction request. Indicates that the current master desires to transfer data using 64-bit transfers.	
	O	Output for the bi-directional 64-bit transaction request.	
		State Meaning	Asserted—The PCI1 controller, as the master of a PCI transaction, desires to use all 64 bits. Negated—The PCI1 controller, as the master of a PCI transaction, uses only 32 bits of the data bus in servicing a data transfer. In PCI host mode, PCI1_REQ64 is asserted (low) immedietly after HRESET negation and remains asserted until one PCI clock cycle after the negation of PCI1_RESET_OUT. In PCI agent mode, if PCI1_REQ64 is sampled high on PORESET negation, the high order bits of the PCI bus (PCI1_AD[63:32], PCI1_C/BE[7:4] and PCI1_PAR64) will never be used.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I	Input for the bi-directional 64-bit transaction request.	
		State Meaning	Asserted—The master of a PCI transaction is requesting to use the full 64-bit data bus for the data phase of the transaction. Negated—The master of a PCI transaction uses only 32 bits of the data bus during the data phase of the transaction.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
PCI1_ACK64	I/O	PCI1 acknowledge 64-bit transfer. Indicates whether current target supports 64-bit transfers during the data phase of the current transaction. This signal is not used if PCI1_64 is 0.	
	O	Output for the bi-directional 64-bit transaction acknowledge.	
		State Meaning	Asserted—The PCI1 controller, as the target of a PCI transaction, may use the full 64-bit data bus for the data phase of the transaction. Negated—The PCI1 controller, as the target of a PCI transaction, may use only 32 bits of the data bus in servicing a data transfer.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I	Input for the bi-directional 64-bit transaction acknowledge.	
		State Meaning	Asserted—The target of a PCI transaction may use the full 64-bit data bus for the data phase of the transaction. Negated—The target of a PCI transaction may only use 32 bits of the data bus during the data phase of the transaction.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3

Table 13-3. PCI1 and PCI2 Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
PCI*n*_RESET_OUT	O	PCI*n* reset. This signal is used only in host mode. It should be left unconnected in agent mode.	
		State Meaning	Asserted—Devices on the PCI bus are in reset. Negated—Devices on the PCI bus operate normally.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
PCI*n*_SERR	I/O	PCI*n* system error	
	O	Outputs for the bi-directional system error.	
		State Meaning	Asserted—An address parity error, a target-abort (when this PCI*n* controller is acting as the initiator), or some other system error (where the result is a catastrophic error) was detected. Negated—No error.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I	Inputs for the bi-directional system error.	
		State Meaning	Asserted—A device (other than this PCI*n* controller) has detected a catastrophic error. Negated—No error.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
PCI*n*_STOP	I/O	PCI*n* stop.	
	O	Outputs for the bi-directional stop.	
		State Meaning	Asserted—The PCI*n* controller, acting as a PCI target, is requesting that the initiator stop the current transaction. Negated—The current transaction can continue.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I	Inputs for the bi-directional stop.	
		State Meaning	Asserted—A target is requesting that this PCI controller, as the initiator, stop the current transaction. Negated—The current transaction can continue.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3

Table 13-3. PCI1 and PCI2 Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
PCI*n*_TRDY	I/O	PCI*n* target ready.	
	O	Outputs for the bi-directional target ready.	
		State Meaning	Asserted—The PCI*n* controller, acting as a PCI target, can complete the current data phase of a PCI transaction. During a read, this PCI*n* controller asserts PCI_TRDY to indicate that valid data is present on PCI_AD[31:0]. During a write, this PCI*n* controller asserts PCI_TRDY to indicate that it is prepared to accept data. Negated—The PCI initiator needs to wait before this PCI*n* controller, acting as a PCI target, can complete the current data phase. During a read, this PCI*n* controller negates PCI_TRDY to insert a wait cycle when it cannot provide valid data to the initiator. During a write, this PCI*n* controller negates PCI_TRDY to insert a wait cycle when it cannot accept data from the initiator.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I	Inputs for the bi-directional target ready.	
		State Meaning	Asserted—Another PCI target is able to complete the current data phase of a transaction. Negated—A wait cycle from another target.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3

13.3 Memory Map/Register Definitions

The PCI controller has the following types of registers:

- The PCI configuration access registers. Used for generating PCI configuration accesses from the CSB. These registers, listed in Table 13-4, are memory-mapped on the CSB and accessed through the IMMR window.
- The PCI memory-mapped registers. Used to manage error functions, general control and status, and address translation control for the inbound path. These registers are shown in Table 13-5. They can be accessed by PCI masters via the PCI controller to the CSB through the PIMMR inbound window. Note that Table 13-5 does not list outbound address translation registers; these are contained in the I/O sequencer (IOS) memory-mapped registers. See Chapter 11, “Sequencer,” for more information.
- The PCI configuration space registers. Defined by the PCI specification. These registers are accessed by PCI masters using configuration accesses and are described in Section 13.3.3, “PCI Configuration Space Registers.”

Table 13-4. PCI Configuration Access Registers

Offset	Use	Access	Section/Page
PCI1 Configuration Access Registers			
0x8300	PCI1_CONFIG_ADDRESS	W	13.3.1.1/13-16
0x8304	PCI1_CONFIG_DATA	R/W	13.3.1.2/13-18
0x8308	PCI1_INT_ACK	R	13.3.1.3/13-18

Table 13-4. PCI Configuration Access Registers (continued)

Offset	Use	Access	Section/Page
PCI2 Configuration Access Registers			
0x8380	PCI2_CONFIG_ADDRESS	W	13.3.1.1/13-16
0x8384	PCI2_CONFIG_DATA	R/W	13.3.1.2/13-18
0x8388	PCI2_INT_ACK	R	13.3.1.3/13-18

Table 13-5. PCI Memory-Mapped Registers

Offset	Use	Access	Section/Page
PCI1 Error Management Registers			
0x8500	PCI error status register (PCI_ESR)	w1c	13.3.2.1/13-18
0x8504	PCI error capture disable register (PCI_ECDR)	R/W	13.3.2.2/13-19
0x8508	PCI error enable register (PCI_EER)	R/W	13.3.2.3/13-20
0x850C	PCI error attributes capture register (PCI_EATCR)	R/W	13.3.2.4/13-21
0x8510	PCI error address capture register (PCI_EACR)	R	13.3.2.5/13-23
0x8514	PCI error extended address capture register (PCI_EEACR)	R	13.3.2.6/13-23
0x8518	PCI error data low capture register (PCI_EDLCR)	R/W	13.3.2.7/13-24
0x851C	PCI error data high capture register (PCI_EDHCR)	R/W	13.3.2.4/13-21
PCI1 Control and Status Registers			
0x8520	PCI general control register (PCI_GCR)	R/W	13.3.2.9/13-24
0x8524	PCI error control register (PCI_ECR)	R/W	13.3.2.10/13-25
0x8528	PCI general status register (PCI_GSR)	R	13.3.2.11/13-26
PCI1 Inbound ATU Registers			
0x8538	PCI inbound translation address register 2 (PITAR2)	R/W	13.3.2.12/13-26
0x853C	Reserved	—	—
0x8540	PCI inbound base address register 2 (PIBAR2)	R/W	13.3.2.13/13-27
0x8544	PCI inbound extended base address register 2 (PIEBAR2)	R/W	13.3.2.14/13-27
0x8548	PCI inbound window attributes register 2 (PIWAR2)	R/W	13.3.2.15/13-28
0x8550	PCI inbound translation address register 1 (PITAR1)	R/W	13.3.2.12/13-26
0x8554	Reserved	—	—
0x8558	PCI inbound base address register 1 (PIBAR1)	R/W	13.3.2.13/13-27
0x855C	PCI inbound extended base address register 1 (PIEBAR1)	R/W	13.3.2.14/13-27
0x8560	PCI inbound window attributes register 1 (PIWAR1)	R/W	13.3.2.15/13-28
0x8568	PCI inbound translation address register 0 (PITAR0)	R/W	13.3.2.12/13-26
0x856C	Reserved	—	—

Table 13-5. PCI Memory-Mapped Registers (continued)

Offset	Use	Access	Section/Page
0x8570	PCI inbound base address register 0 (PIBAR0)	R/W	13.3.2.13/13-27
0x8578	PCI inbound window attributes register 0 (PIWAR0)	R/W	13.3.2.14/13-27
0x857C–0x85FF	Reserved	—	—
PCI2 Memory-Mapped Registers			
0x8600–0x86FF	PCI2 registers **Note:** The PCI2 interface has the same memory-mapped registers that are described for PCI1 from 0x8500 to 0x85FF except the offsets are from 0x8600 to 0x86FF.		

13.3.1 PCI Configuration Access Registers

This section describes the registers used to allow a local bus master to access the PCI configuration space, and generate special cycle or interrupt acknowledge transactions on the PCI bus. A special case provides access to the PCI controller's internal PCI configuration registers. The PCI registers, PCI_CONFIG_ADDRESS, PCI_CONFIG_DATA, and PCI_INT_ACK, are little endian registers.

13.3.1.1 PCI_CONFIG_ADDRESS

Figure 13-3 shows the PCI_CONFIG_ADDRESS register fields.

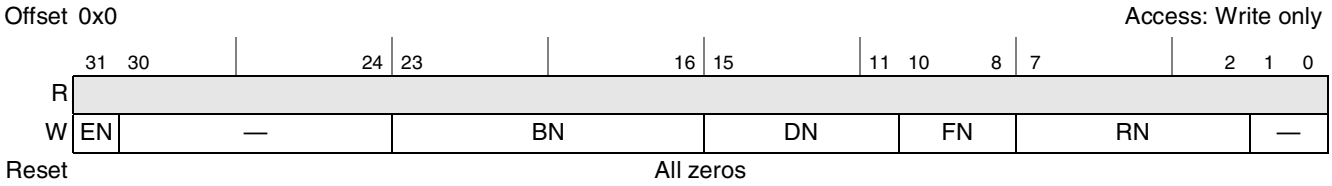


Figure 13-3. PCI_CONFIG_ADDRESS Register

The PCI_CONFIG_ADDRESS register holds the address for an access to the PCI configuration space from the local bus. This register must be programmed before accessing PCI_CONFIG_DATA to perform the transaction. Only 32-bit accesses are permitted.

If EN=1, BN=0, and DN=0, the access is to the internal PCI configuration registers, so no transaction is generated on the PCI bus.

If EN=1, BN=0, DN=31, FN=7, and RN=0, writing to PCI_CONFIG_DATA generates a special cycle transaction and reading from PCI_CONFIG_DATA generates an interrupt acknowledge transaction.

Table 13-6 shows the bit settings of the PCI_CONFIG_ADDRESS register.

Table 13-6. PCI_CONFIG_ADDRESS Field Descriptions

Bits	Name	Description
31	EN	Enable configuration transaction. Determines the type of transaction to be generated. 0 No configuration transaction will be generated by accessing the CONFIG_DATA register. Such an access will be passed through to the PCI bus as an I/O transaction. Since this is generally not desirable, the user should not access CONFIG_DATA when the EN bit is 0. 1 A configuration transaction will be generated by accessing the CONFIG_DATA register if BN and DN are not both zero.
30–24	—	Reserved
23–16	BN	Bus number. Specifies the bus segment to which a configuration transaction is directed. If this field is 0, a Type 0 configuration transaction is generated. Otherwise, a Type 1 configuration transaction is generated.
15–11	DN	Device number. Specifies the device to which a configuration transaction is directed. For a Type 0 configuration transaction, this field is decoded to individual PCI1_IDSEL signals for the address phase according to the following values. For a Type 1 configuration transaction, this field is used directly for the address phase. **Value** — **AD Signal that is Driving High** 01010 — 31 01011 — 11 01100 — 12 01101 — 13 01110 — 14 01111 — 15 10000 — 16 10001 — 17 10010 — 18 10011 — 19 10100 — 20 10101 — 21 10110 — 22 10111 — 23 11000 — 24 11001 — 25 11010 — 26 11011 — 27 11100 — 28 11101 — 29 11110 — 30 11111 — Special cycle / interrupt acknowledge 00000 — Internal access Others — Reserved
10–8	FN	Function number. Specifies the function to which the configuration transaction is directed on a multi-function device. It is used directly in the address phase of the configuration transaction.

Table 13-6. PCI_CONFIG_ADDRESS Field Descriptions (continued)

Bits	Name	Description
7–2	RN	Register number. Specifies the register being accessed in the PCI configuration space.
1–0	—	Reserved

13.3.1.2 PCI_CONFIG_DATA

An access to PCI_CONFIG_DATA usually generates a PCI configuration transaction if PCI_CONFIG_ADDRESS[EN] is set. There are some exceptions contained in the description of PCI_CONFIG_ADDRESS[EN].

This register may be accessed with an 8-, 16-, or 32-bit access, depending on the width of the register targeted by the configuration transaction.

Figure 13-4 shows the PCI_CONFIG_DATA register fields.

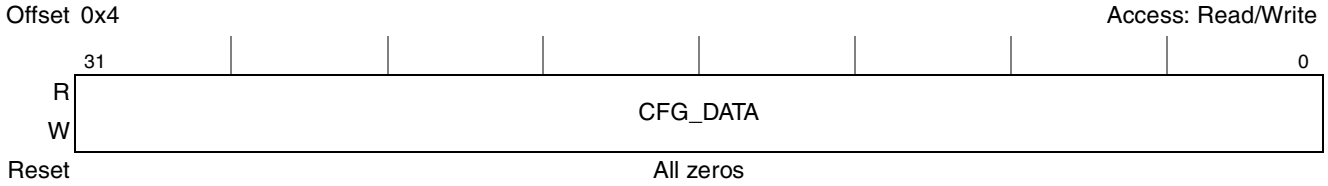


Figure 13-4. PCI_CONFIG_DATA

Table 13-7 shows the bit settings of the PCI_CONFIG_DATA register.

Table 13-7. PCI_CONFIG_DATA Field Descriptions

Bits	Name	Description
31–0	CFG_DATA	Configuration data. This field contains the data transferred on a PCI configuration transaction.

13.3.1.3 PCI Interrupt Acknowledge Register (PCI_INT_ACK)

Reading this register generates an interrupt acknowledge transaction on the PCI bus. The value that is read is undefined.

13.3.2 PCI Memory-Mapped Control and Status Registers

This section describes the control and status registers.

13.3.2.1 PCI Error Status Register (PCI_ESR)

The PCI error status register (PCI_ESR) contains status bits for various types of error conditions captured by the PCI controller. Each status bit is set when the corresponding error condition is captured. PCI_ESR is a write-1-to-clear type register. A bit is cleared whenever the register is written and the data in the corresponding bit location is a 1. Figure 13-5 shows the PCI_ESR fields.

Offset 0x00 Access: w1c

	0	1	—	20	21	22	23	24	25	26	27	—	31
R	MERR				APAR	PCISERR	MPERR	TPERR	NORSP	TABT			
W	w1c				w1c	w1c	w1c	w1c	w1c	w1c			
Reset	All zeros												

Figure 13-5. PCI Error Status Register (PCI_ESR)

Table 13-8 describes the bit settings of the PCI_ESR register.

Table 13-8. PCI_ESR Field Descriptions

Bits	Name	Description
0	MERR	Multiple errors. Set if any other bit of this register is 1 and the same error type occurs again.
1–20	—	Reserved
21	APAR	Address parity error. Set when there is an address parity error on a PCI access initiated by a device other than this PCI controller.
22	PCISERR	PCI system error.Set when the $\overline{\text{PCI_SERR}}$ input signal is asserted. See Table 13-3 for more information on $\overline{\text{PCI_SERR}}$.
23	MPERR	Master parity error. Set when the $\overline{\text{PCI_PERR}}$ input signal is asserted on a write access initiated by this PCI controller or when a data parity error is detected by this PCI controller on a read access that it initiated.
24	TPERR	Target parity error. Set when this PCI controller is the target of a transaction and the $\overline{\text{PCI_PERR}}$ input signal is asserted on a read access or a data parity error is detected by this PCI controller on a write access.
25	NORSP	No response. Set when there is no response to a transaction initiated by this PCI controller on the PCI bus (no $\overline{\text{PCI}n\text{_DEVSEL}}$ assertion).
26	TABT	Target abort. Set when a PCI target abort occurs on a transaction initiated by this PCI controller.
27–31	—	Reserved

13.3.2.2 PCI Error Capture Disable Register (PCI_ECDR)

PCI_ECDR contains fields for controlling the capture of the transaction that caused an error. Each bit corresponds to the error condition reported in the PCI error status register (PCI_ESR). Note that only the first error is captured, so disabling the capture of some error types may allow greater visibility of the significant errors.

- 1 = Do not capture the transaction that caused this error.
- 0 = Capture the transaction that caused this error.

Figure 13-6 shows the PCI_ECDR fields.

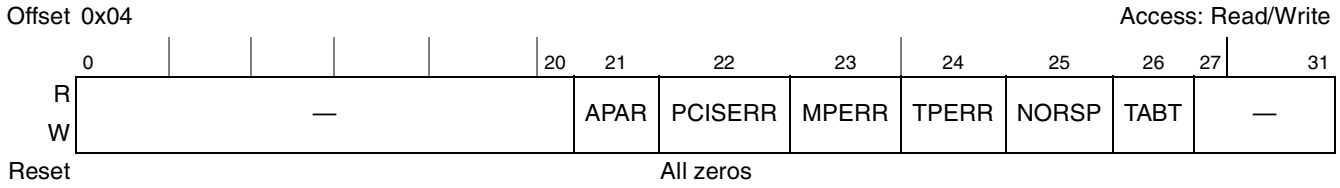


Figure 13-6. PCI Error Capture Disable Register (PCI_ECDR)

Table 13-9 describes the bit settings of the PCI_ECDR register.

Table 13-9. PCI_ECDR Field Descriptions

Bits	Name	Description
0–20	—	Reserved
21	APAR	Address parity error. Disable capture for address parity errors
22	PCISERR	PCI system error. Disable capture for received $\overline{\text{PCI_SERR}}$ errors
23	MPERR	Master parity error. Disable capture for master $\overline{\text{PCI_PERR}}$ errors
24	TPERR	Target parity error. Disable capture for target $\overline{\text{PCI_PERR}}$ errors
25	NORSP	No response. Disable capture for master-abort errors
26	TABT	Target abort. Disable capture for target abort errors
27–31	—	Reserved

13.3.2.3 PCI Error Enable Register (PCI_EER)

PCI_EER contains fields for enabling the assertion of an interrupt for the error conditions reported in the PCI error status register (PCI_ESR).

- 1 = The interrupt is enabled.
- 0 = The interrupt is disabled.

Figure 13-7 shows the PCI_EER fields.

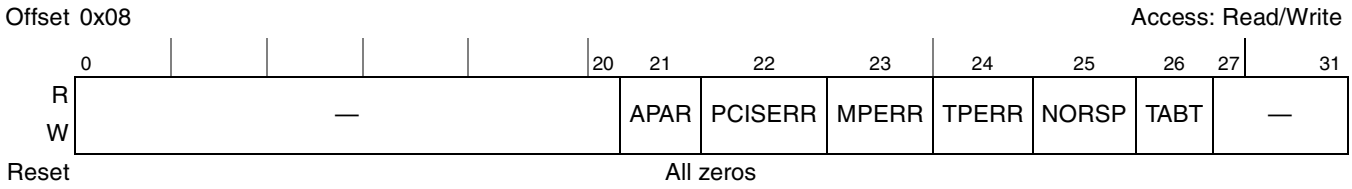


Figure 13-7. PCI Error Enable Register (PCI_EER)

Table 13-10 describes the bit settings of the PCI_EER register.

Table 13-10. PCI_EER Field Descriptions

Bits	Name	Description
0–20	—	Reserved
21	APAR	Address parity error. Generate an interrupt when the corresponding bit of the PCI_ESR is 1.
22	PCISERR	PCI system error. Generate an interrupt when the corresponding bit of the PCI_ESR is 1.
23	MPERR	Master parity error. Generate an interrupt when the corresponding bit of the PCI_ESR is 1.
24	TPERR	Target parity error. Generate an interrupt when the corresponding bit of the PCI_ESR is 1.
25	NORSP	No response. Generate an interrupt when the corresponding bit of the PCI_ESR is 1.
26	TABT	Target abort. Generate an interrupt when the corresponding bit of the PCI_ESR is 1.
27–31	—	Reserved

13.3.2.4 PCI Error Attributes Capture Register (PCI_EATCR)

PCI_EATCR contains fields for storing information associated with the first PCI error captured. Figure 13-8 shows the PCI_EATCR fields.

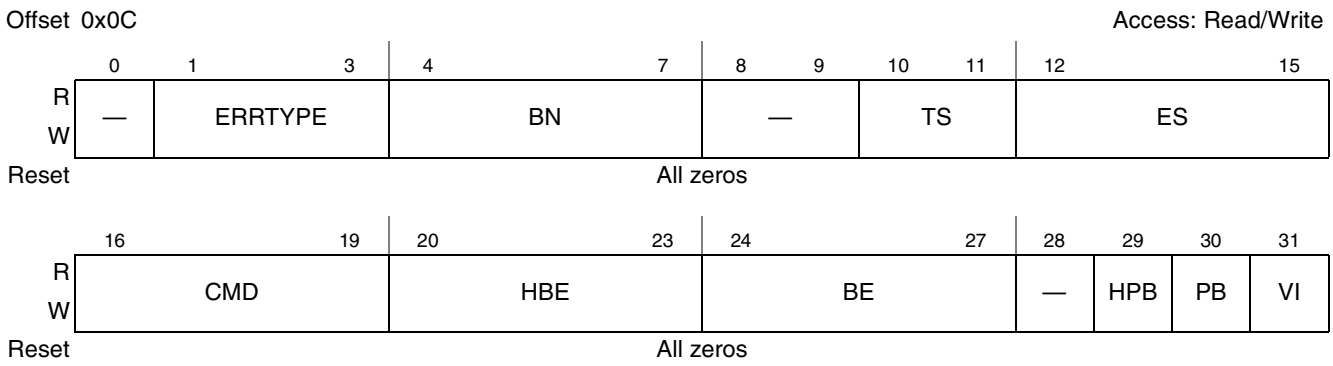


Figure 13-8. PCI Error Attributes Capture Register (PCI_EATCR)

Table 13-11 describes the bit settings of the PCI_EATCR register.

Table 13-11. PCI_EATCR Field Descriptions

Bits	Name	Description
0	—	Reserved
1–3	ERRTYPE	First error type. This field is encoded to indicate the type of the first PCI error captured. 000 Address parity error 001 Write data parity error 010 Read data parity error 011 Master abort 100 Target abort 101 System error indication received 110 Parity error indication received on a read 111 Parity error indication received on a write

Table 13-11. PCI_EATCR Field Descriptions (continued)

Bits	Name	Description
4–7	BN	Beat number. This field provides the data beat number on which the error occurred for data parity errors. The value of this field is undefined for other error types. The beat values are described as follows: 0000 1st beat 0001 2nd beat 0010 3rd beat 0011 4th beat 0100 5th beat 0101 6th beat 0110 7th beat 0111 8th beat 1000 9th beat or beyond (transaction larger than one cache line) Others Reserved
8–9	—	Reserved
10–11	TS	Transaction size. Indicates the size of the transaction in units of doublewords (8 bytes). If the transaction crossed a cache line (32-byte) boundary, this field indicates the number of actual double words in the cache line on which the error occurred. This field is valid only if the PCI controller was the master of the transaction. 00 4 double words 01 1 double word 10 2 double words 11 3 double words
12–15	ES	Error source. This field indicates the source of the PCI transaction. 0000 External master 0101 DMA Others reserved
16–19	CMD	PCI command. Contains the PCI command PCI_CBE[3:0] of the transaction.
20–23	HBE	PCI high byte enables. Contains the PCI high byte enables for the data word.
24–27	BE	PCI byte enables. Contains the PCIlow byte enables PCI_CBE[3:0] for the data word.
28	—	Reserved
29	HPB	High parity bit. Contains the high PCI parity bit for the data word.
30	PB	Parity bit. Contains thelow PCI parity bit for the captured data word.
31	VI	Error information valid. This bit indicates that the error information captured in this register, PCI_EACR, PCI_EEACR, PCI_EDHCR, and PCI_EDLCR is valid. 0 No valid error information 1 Error information is valid

13.3.2.5 PCI Error Address Capture Register (PCI_EACR)

PCI_EACR contains fields for storing the low portion of the address associated with the first PCI error captured. Figure 13-9 shows the PCI_EACR fields.

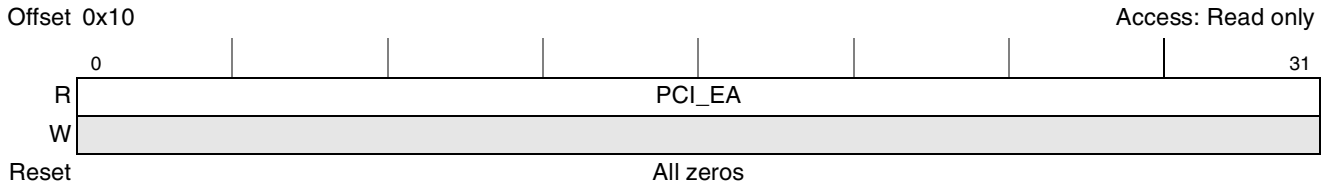


Figure 13-9. PCI Error Address Capture Register (PCI_EACR)

Table 13-12 describes the bit settings of the PCI_EACR register.

Table 13-12. PCI_EACR Field Description

Bits	Name	Description
0–31	PCI_EA	PCI error address. Contains the low portion of the address associated with the first detected error. Read only.

13.3.2.6 PCI Error Extended Address Capture Register (PCI_EEACR)

PCI_EEACR contains fields for storing the high portion of the address associated with the first PCI error captured. Figure 13-10 shows the PCI_EEACR fields.

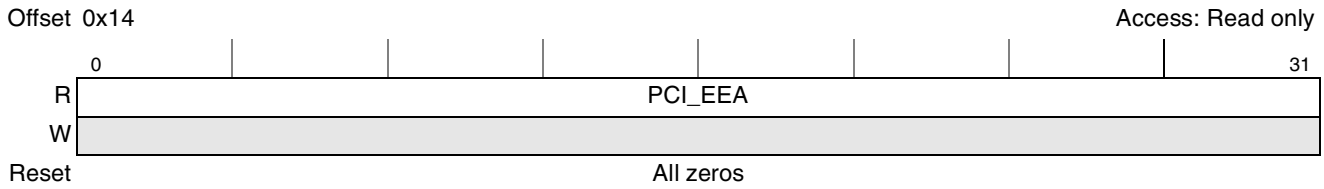


Figure 13-10. PCI Error Extended Address Capture Register (PCI_EEACR)

Table 13-13 describes the bit settings of the PCI_EEACR register.

Table 13-13. PCI_EEACR Field Description

Bits	Name	Description
0–31	PCI_EEA	PCI error extended address. Contains the high portion of the address associated with the first detected error.

13.3.2.7 PCI Error Data Low Capture Register (PCI_EDLCR)

PCI_EDLCR contains fields for storing the data associated with the first PCI error captured. Figure 13-11 shows the PCI_EDLCR fields.

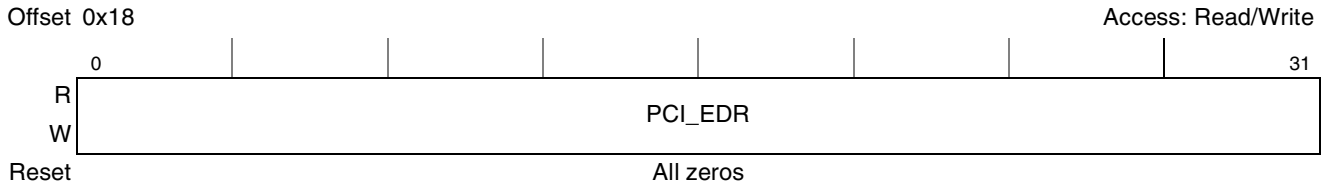


Figure 13-11. PCI Error Data Low Capture Register (PCI_EDLCR)

Table 13-14 describes the bit settings of the PCI_EDLCR register.

Table 13-14. PCI_EDLCR Field Description

Bits	Name	Description
0–31	PCI_EDR	PCI error data. Contains the data associated with the first detected error.

13.3.2.8 PCI Error Data High Capture Register (PCI_EDHCR)

PCI_EDHCR contains fields for storing the data associated with the first PCI error captured. This register is not used if PCI_64 is 0. Figure 13-12 shows the PCI_EDHCR fields.

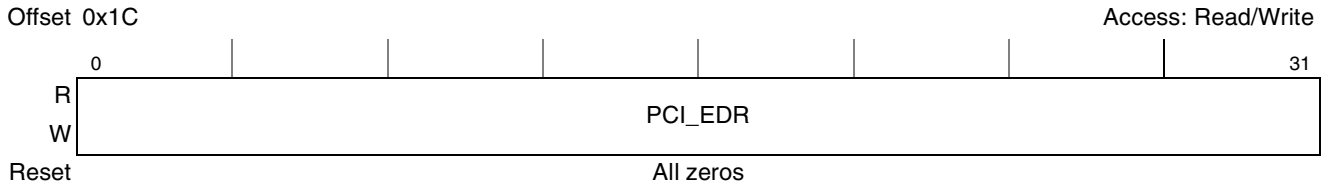


Figure 13-12. PCI Error Data High Capture Register (PCI_EDHCR)

Table 13-15 describes the bit settings of the PCI_EDHCR register.

Table 13-15. PCI_EDLCR Field Description

Bits	Name	Description
0–31	PCI_EDR	PCI error data. Contains the data associated with the first detected error.

13.3.2.9 PCI General Control Register (PCI_GCR)

PCI_GCR contains fields for controlling the behavior of the internal arbiter, the state of the bus signals, and the PCI reset signal for host mode. Figure 13-13 shows the PCI_GCR fields.

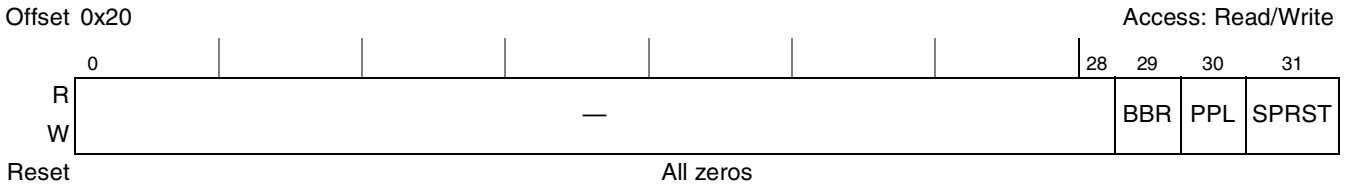


Figure 13-13. PCI General Control Register (PCI_GCR)

Table 13-16 shows the bit settings of PCI_GCR. The bits that are not reserved have read and write capability.

Table 13-16. PCI_GCR Field Descriptions

Bits	Name	Description
0–28	—	Reserved
29	BBR	Block bus requests. This bit could be used to prepare for entering a low-power mode by preventing transactions on the PCI bus. 0 External bus requests are treated normally. 1 Block external bus requests. When this bit is set, all bus requests from external devices to the PCI controller's internal arbiter are blocked, and the bus is continuously granted to the PCI controller.
30	PPL	PCI pins low. This bit could be used to put the bus signals in a safe electrical state when the devices on the bus are powered down. This bit should never be set during normal operation of the PCI bus. 0 PCI pins function normally 1 PCI pins in the low state. Setting this bit forces all the output and bidirectional pins of the PCI bus to be driven low.
31	SPRST	Soft PCI reset. This bit provides software control of the $\overline{\text{PCI_RESET_OUT}}$ output signal. It is only valid in host mode. 0 $\overline{\text{PCI_RESET_OUT}}$ is driven low. 1 $\overline{\text{PCI_RESET_OUT}}$ is driven high.

13.3.2.10 PCI Error Control Register (PCI_ECR)

PCI_ECR contains fields for determining whether an interrupt or machine check is generated for the error conditions reported in the PCI error status register (PCI_ESR). Note that if the corresponding bit in the PCI error enable register (PCI_EER) is clear, the bit in the PCI error control register (PCI_ECR) has no effect.

1 = A machine check is generated.
0 = An interrupt is generated.

Figure 13-14 shows the PCI_ECR fields.

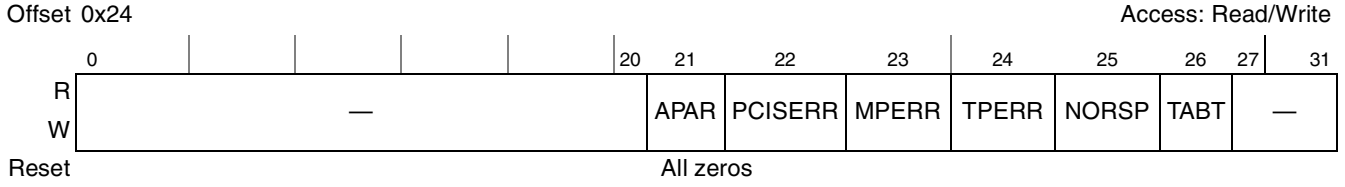


Figure 13-14. PCI Error Control Register (PCI_ECR)

Table 13-17 describes the bit settings of the PCI_ECR register.

Table 13-17. PCI_ECR Field Descriptions

Bits	Name	Description
0–20	—	Reserved
21	APAR	0 An interrupt is generated if the corresponding bit of the PCI_ESR is 1. 1 A machine check is generated if the corresponding bit of the PCI_ESR is 1.
22	PCISERR	0 An interrupt is generated if the corresponding bit of the PCI_ESR is 1. 1 A machine check is generated if the corresponding bit of the PCI_ESR is 1.
23	MPERR	0 An interrupt is generated if the corresponding bit of the PCI_ESR is 1. 1 A machine check is generated if the corresponding bit of the PCI_ESR is 1.
24	TPERR	0 An interrupt is generated if the corresponding bit of the PCI_ESR is 1. 1 A machine check is generated if the corresponding bit of the PCI_ESR is 1.
25	NORSP	0 An interrupt is generated if the corresponding bit of the PCI_ESR is 1. 1 A machine check is generated if the corresponding bit of the PCI_ESR is 1.
26	TABT	0 An interrupt is generated if the corresponding bit of the PCI_ESR is 1. 1 A machine check is generated if the corresponding bit of the PCI_ESR is 1.
27–31	—	Reserved

13.3.2.11 PCI General Status Register (PCI_GSR)

PCI_GSR contains fields for providing status information, shown in Figure 13-15.

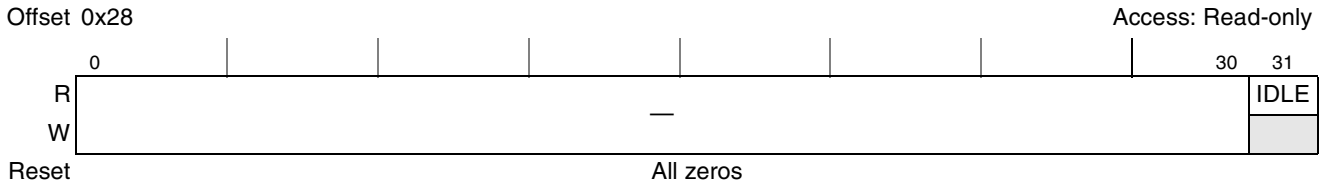


Figure 13-15. PCI General Status Register (PCI_GSR)

Table 13-18 shows the bit settings of the PCI_GSR register. All bits are read-only.

Table 13-18. PCI_GSR Field Descriptions

Bits	Name	Description
0–30	—	Reserved
31	IDLE	PCI controller is idle. Indicates when the PCI bus is totally idle before setting PCI_GCR[PPL]. 0 The PCI controller is active. 1 The PCI controller is idle.

13.3.2.12 PCI Inbound Translation Address Registers (PITAR*n*)

PITAR*n* contains fields for defining the starting point of the inbound translation windows in the local memory space. See Chapter 11, “Sequencer for more information on outbound address translation registers.

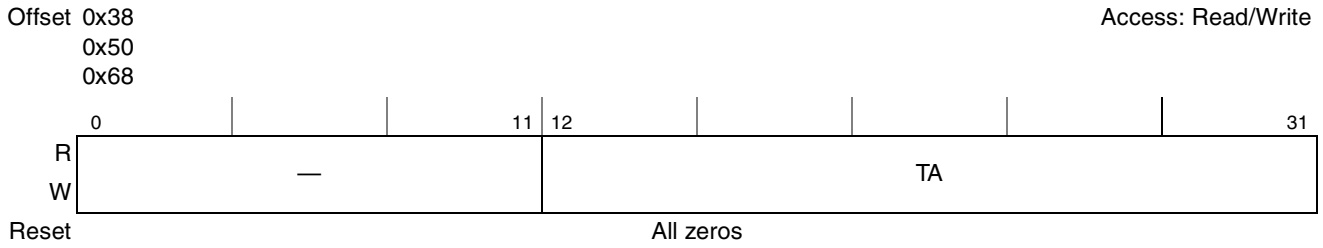


Figure 13-16. PCI Inbound Translation Address Registers (PITAR*n*)

Table 13-19 shows the bit settings of PITAR*n*.

Table 13-19. PITAR*n* Field Descriptions

Bits	Name	Description
0–11	—	Reserved
12–31	TA	Translation address. Contains the starting address of the inbound translated address. TA corresponds to the 20 highest-order bits of a 32-bit local address.

13.3.2.13 PCI Inbound Base Address Registers (PIBAR*n*)

PIBAR*n* contains fields for defining the starting point of the inbound windows in the PCI memory space. A write to a PIBAR*n* register also causes a change in the base address bits in the corresponding GPL base address register in the PCI configuration space. Figure 13-17 shows the PIBARx fields.

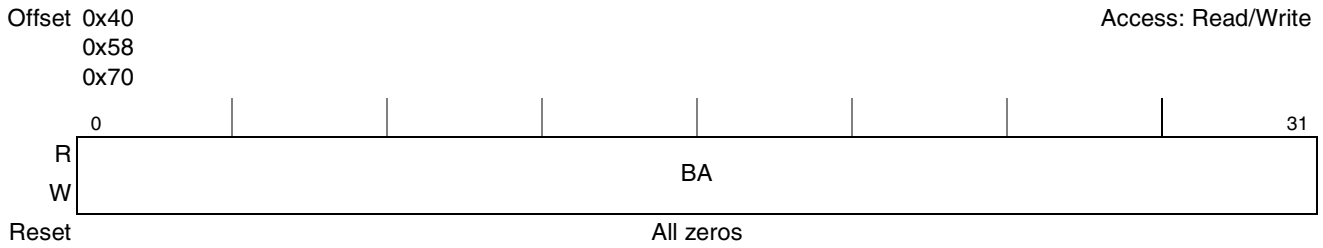


Figure 13-17. PCI Inbound Base Address Registers (PIBAR*n*)

Table 13-20 shows the bit settings of PIBAR*n*.

Table 13-20. PIBAR*n* Field Descriptions

Bits	Name	Description
0–31	BA	Base address. Contains the starting address in the PCI memory space of the inbound window. This field corresponds to bits 43–12 of a 64-bit address. In PIBAR0, the upper 12 bits are reserved because only a 32-bit address is supported.

13.3.2.14 PCI Inbound Extended Base Address Registers (PIEBAR*n*)

PIEBAR*n* contains fields for defining the high portion of the starting point of the inbound windows in the PCI memory space. Figure 13-18 shows the PIEBAR*n* fields.

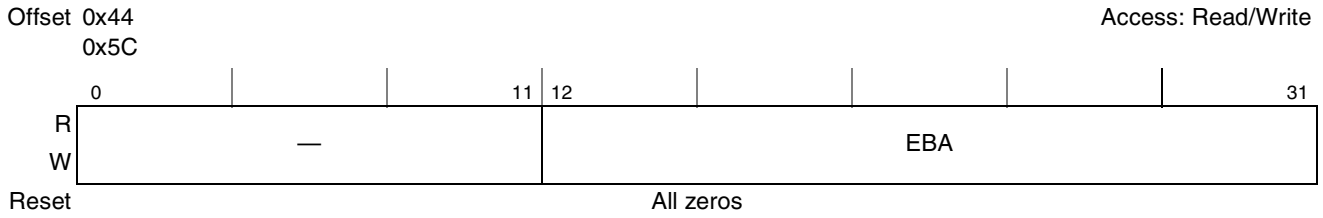


Figure 13-18. PCI Inbound Extended Base Address Registers (PIEBAR*n*)

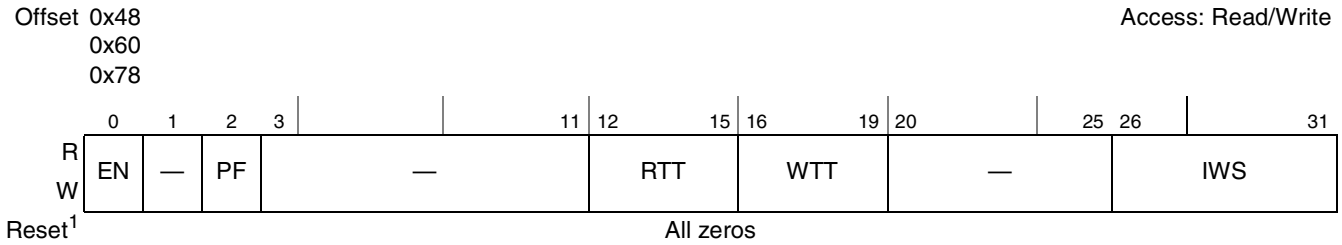
Table 13-21 shows the bit settings of PIEBAR*n*.

Table 13-21. PIEBAR*n* Field Descriptions

Bits	Name	Description
0–11	—	Reserved
12–31	EBA	Extended base address. Contains the high portion of the starting address in the PCI memory space of the inbound base address. This 20-bit field corresponds to bits 63–44 of a 64-bit address.

13.3.2.15 PCI Inbound Window Attribute Registers (PIWAR*n*)

PIWAR*n* contains fields for defining the size of an inbound translation window. It also defines some properties of the window. Figure 13-19 shows the PIWAR*n* fields. See Section 4.3.1.1, "Reset Configuration Word Source," for more information on reset configuration.



1 When using a hard-coded reset configuration option, the reset value of this register is 0x80055013, which enables a 1 MB window.

Figure 13-19. PCI Inbound Window Attribute Registers (PIWAR*n*)

Table 13-22 shows the bit settings of PIWAR*n*.

Table 13-22. PIWAR*n* Field Descriptions

Bits	Name	Description
0	EN	Enable. Used to enable the address translation window. 0 Address translation is disabled for this window. 1 Address translation is enabled for this window. PCI addresses that match the definition of the window will be recognized by the PCI controller and translated to the local memory space.
1	—	Reserved

Table 13-22. PIWAR*n* Field Descriptions (continued)

Bits	Name	Description
2	PF	Prefetchable. Defines whether the transactions that are translated through this window are prefetchable on the local bus. Streaming the transactions requires the memory space to be prefetchable. 0 Not prefetchable 1 Prefetchable
3–11	—	Reserved
12–15	RTT	Read transaction type. Determines the type of transaction performed on the local bus when the PCI transaction is a read. The RTT values are described as follows: 0100 Read without snoop on system bus 0101 Read with snoop on system bus Others reserved
16–19	WTT	Write transaction type. Determines the type of transaction performed on the local bus when the PCI transaction is a write. The WTT values are described as follows: 0100 Write without snoop of local processor 0101 Write with snoop of local processor Others reserved
20–25	—	Reserved
26–31	IWS	Inbound window size. Indicates the size of the inbound translation window. Inbound translation window size N which is the encoded 2^(N+1) bytes window size. The smallest window is 4 Kbytes (N = 11) 000000–001010 Reserved 001011 4-Kbyte window size 001100 8-Kbyte window size ... 011110 2-Gbyte window size 011111–111111 Reserved

13.3.3 PCI Configuration Space Registers

This section describes the PCI configuration space registers. These registers are shown with descending bit numbering to correspond to the PCI standard.

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.

Table 13-23 shows the PCI configuration registers that are mapped in PCI configuration space. Some fields are common to registers in both spaces to ensure consistency. These fields are discussed in the register definitions.

Table 13-23. PCI Configuration Space Registers

Address	Use	Access
00	Vendor ID configuration register	R
02	Device ID configuration register	R
04	PCI command configuration register	R/W
06	PCI status configuration register	Read/bit-reset
08	Revision ID configuration register	R
09	Standard programming interface	R
0A	Subclass code configuration register	R
0B	Base class code configuration register	R
0C	Cache line size configuration register	R/W
0D	Latency timer configuration register	R/W
0E	Header type configuration register	R
0F	BIST control configuration register	R
10	PIMMR base address register	R/W
14	GPL base address register 0	R/W
18	GPL base address register 1	R/W
1C	GPL extended base address register 1	R/W
20	GPL base address register 2	R/W
24	GPL extended base address register 2	R/W
2C	Subsystem vendor ID configuration register	R
2E	Subsystem device ID configuration register	R
34	Capabilities pointer configuration register	R
3C	Interrupt line configuration register	R/W
3D	Interrupt pin configuration register	R
3E	Minimum grant configuration register	R
3F	Maximum latency configuration register	R
44	PCI function configuration register	R/W
46	PCI arbiter control register (PCIACR)	R/W
48	Hot swap register block	R/W

13.3.3.1 Vendor ID Configuration Register

Figure 13-20 shows the vendor ID fields. This is a read-only register.

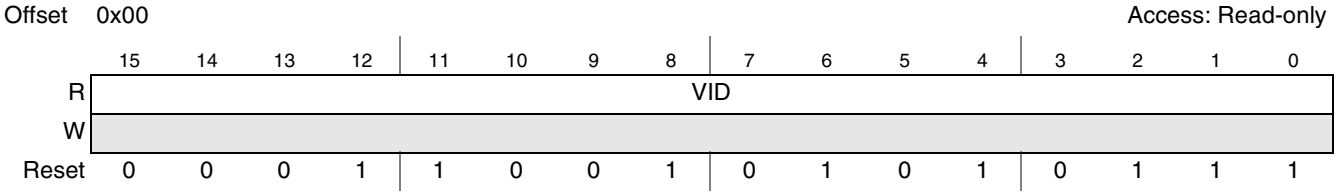


Figure 13-20. Vendor ID Configuration Register

Table 13-24 shows the bit settings of the vendor ID register.

Table 13-24. Vendor ID Configuration Register Field Descriptions

Bits	Name	Description
15–0	VID	Vendor ID. The read-only value 0x1957 specifies Freescale Semiconductor as the manufacturer of the device.

13.3.3.2 Device ID Configuration Register

Figure 13-21 shows the device ID fields. This is a read only register.

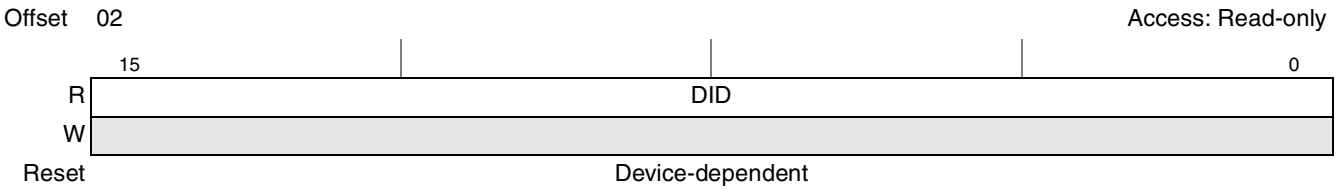


Figure 13-21. Device ID Configuration Register

Table 13-25 shows the bit settings of the device ID register.

Table 13-25. Device ID Configuration Register Field Descriptions

Bits	Name	Description
15–0	DID	Device ID. This field identifies the device. 0080 MPC8349EA TBGA 0081 MPC8349A TBGA 0082 MPC8347EA TBGA 0083 MPC8347A TBGA 0084 MPC8347EA PBGA 0085 MPC8347A PBGA 0086 MPC8343EA PBGA 0087 MPC8343A PBGA

13.3.3.3 PCI Command Configuration Register

Figure 13-22 shows the PCI command fields.

Offset 04 Access: Mixed

	15	11	10	9	8	7	6	5	4	3	2	1	0
R	—		INTD	FB-B	SERREN	—	PERRR	—	MWI	SC	BMST	MEM	I/O
W													
Reset	All zeros												

Figure 13-22. PCI Command Configuration Register

Table 13-26 shows the bit settings of the PCI command register.

Table 13-26. PCI Command Configuration Register Field Descriptions

Bits	Name	Description
15–11	—	Reserved
10	INTD	Interrupt Disable. Setting this bit masks the $\overline{\text{PCI_INTA}}$ output. 0 $\overline{\text{PCI_INTA}}$ provides the device interrupt status. 1 $\overline{\text{PCI_INTA}}$ is always negated.
9	FB-B	Fast back-to-back. Hard-wired to 0.
8	SERREN	SERR enable. This bit is an enable bit for the SERR driver. Address parity errors are reported only if this bit and bit 6 are 1. 0 $\overline{\text{PCI_SERR}}$ is never asserted. 1 $\overline{\text{PCI_SERR}}$ may be asserted to indicate error conditions.
7	—	Reserved
6	PERRR	Parity error response. Controls the PCI controller's response to a parity error. 0 Parity errors are ignored and normal operation continues. 1 Standard parity error treatment.
5	—	Reserved
4	MWI	Memory-write-and-invalidate. Hard-wired to 0.
3	SC	Special cycles. Hard-wired to 0.
2	BMST	Bus master. Controls the PCI controller's ability to be a master on the PCI bus. At reset, this bit is cleared in Agent Mode and set in Host Mode. 0 The PCI controller does not generate PCI accesses. 1 The PCI controller behaves as a bus master.
1	MEM	Memory space. Controls the response to memory space accesses. 0 The PCI controller does not respond to Memory Space accesses. 1 The PCI controller as a target responds to Memory Space accesses.
0	I/O	I/O space. Hard-wired to 0.

13.3.3.4 PCI Status Configuration Register

This register is used to record status information for PCI bus-related events. Some of the bits are hard-wired to indicate the capabilities of the PCI controller. Other bits can be cleared by writing 1 to the bit location. Figure 13-23 shows the PCI status fields.

Offset 06 Access: Mixed

	15	14	13	12	11	10	9	8	7	6	5	4	3	2		0
R	DPERR	SSERR	RMA	RTA	STA	DEVSEL_T		DPD	FB-BC	—	66M	CL	INTS	—		
W	w1c	w1c	w1c	w1c	w1c			w1c					w1c			
Reset	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0

Figure 13-23. PCI Status Configuration Register

Table 13-27 shows the bit settings of the PCI status register.

Table 13-27. PCI Status Configuration Register Field Descriptions

Bits	Name	Description
15	DPERR	Detected parity error. Set whenever the PCI controller detects a parity error on the PCI bus, even if parity error handling is disabled (as controlled by bit 6 in the PCI Command register).
14	SSERR	Signaled system error. Set whenever $\overline{\text{PCI_SERR}}$ is asserted.
13	RMA	Received master abort. Set whenever the PCI controller, acting as the PCI master on the PCI bus, terminates a transaction (except for a special-cycle) using master-abort.
12	RTA	Received target abort. Set whenever a transaction initiated by this PCI controller on the PCI bus is terminated by a target-abort.
11	STA	Signaled target abort. Set whenever the PCI controller, acting as the PCI target on the PCI bus, issues a target-abort to a PCI master.
10–9	DEVSEL_T	DEVSEL timing. Hard-wired to 00.
8	DPD	Master data parity error. Set when a data parity error is detected on the PCI bus, if the PCI controller is the master that initiated the transaction and bit 6 in the PCI command register is set.
7	FB-BC	Fast back-to-back capable. Hard-wired to 1.
6	—	Reserved
5	66M	66-MHz capable. Hard-wired to 1.
4	CL	Capabilities list. Hard-wired to 1.
3	INTS	Interrupt status. Contains the status of the device interrupt. The value of this bit is not affected by the INTD bit of the PCI command configuration register.
2–0	—	Reserved

13.3.3.5 Revision ID Configuration Register

Figure 13-24 shows the revision ID fields.

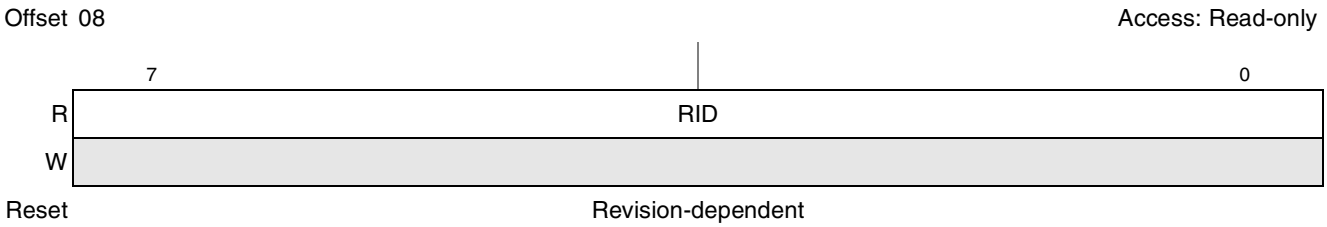


Figure 13-24. Revision ID Configuration Register

Table 13-28 shows the bit settings of the revision ID register.

Table 13-28. Revision ID Configuration Register Field Descriptions

Bits	Name	Description
7–0	RID	Revision ID. Specifies a revision code of the PCI controller. 00000000 Revision 0.0.

13.3.3.6 Standard Programming Interface Configuration Register

Figure 13-25 shows the standard programming interface fields. This is the lower byte of the class code.

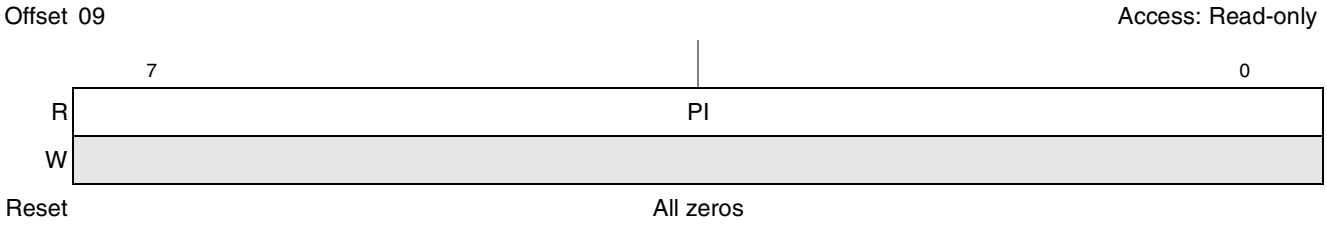


Figure 13-25. Standard Programming Interface Configuration Register

Table 13-29 shows the bit settings of the standard programming interface register.

Table 13-29. Standard Programming Interface Configuration Register Field Descriptions

Bits	Name	Description
7–0	PI	Programming interface. This field is hard-wired to 0x00.

13.3.3.7 Subclass Code Configuration Register

Figure 13-26 shows the subclass code fields. This is the middle byte of the class code.

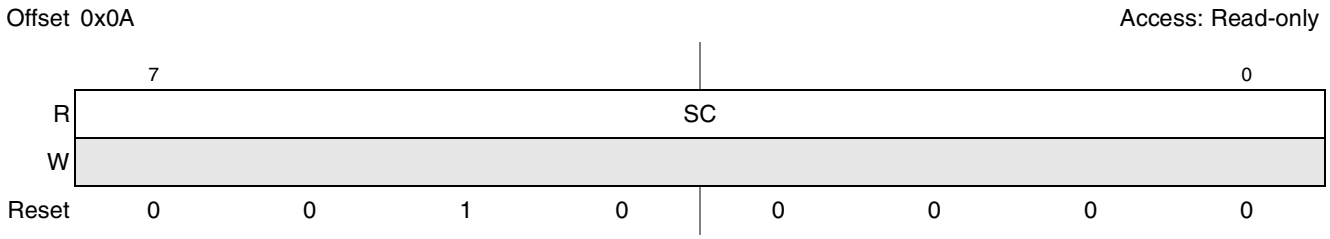


Figure 13-26. Subclass Code Configuration Register

Table 13-30 shows the bit settings of the subclass code register.

Table 13-30. Subclass Code Configuration Register Field Descriptions

Bits	Name	Description
7–0	SC	Sub-class code. This field is hard-wired to 0x20, indicating a Power PC processor.

13.3.3.8 Base Class Code Configuration Register

Figure 13-27 shows the base class code fields. This is the upper byte of the class code.

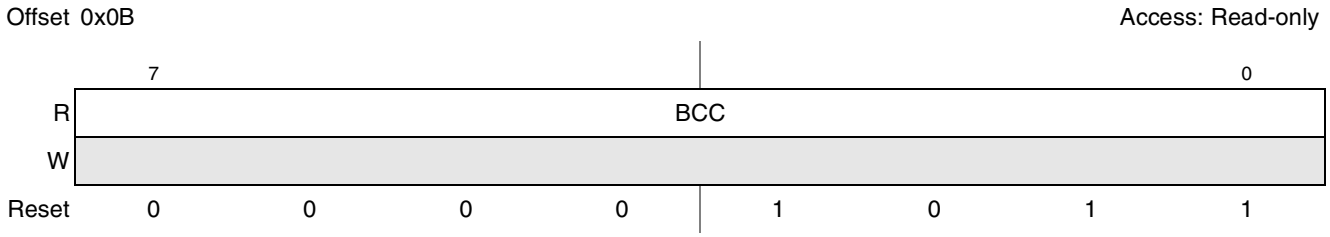


Figure 13-27. Base Class Code Configuration Register

Table 13-31 shows the bit settings of the class code register.

Table 13-31. Class Code Configuration Register Field Descriptions

Bits	Name	Description
7–0	BCC	Base class code. This field is hard-wired to 0x0B, indicating a processor.

13.3.3.9 Cache Line Size Configuration Register

Figure 13-28 shows the cache line size fields.

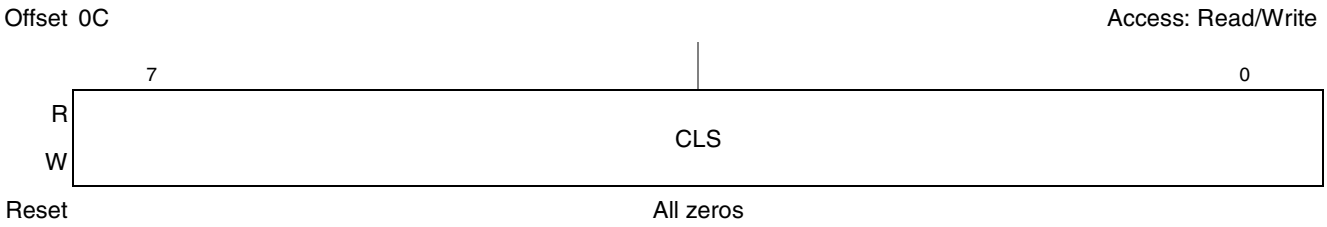


Figure 13-28. Cache Line Size Configuration Register

Table 13-32 shows the bit settings of the cache line size register.

Table 13-32. Cache Line Size Configuration Register Field Descriptions

Bits	Name	Description
7–0	CLS	Cache line size. Cache-line in terms of 32-bit words. Although the register is writable, only the value 0x08 is legal.

13.3.3.10 Latency Timer Configuration Register

Figure 13-29 shows the latency timer fields.

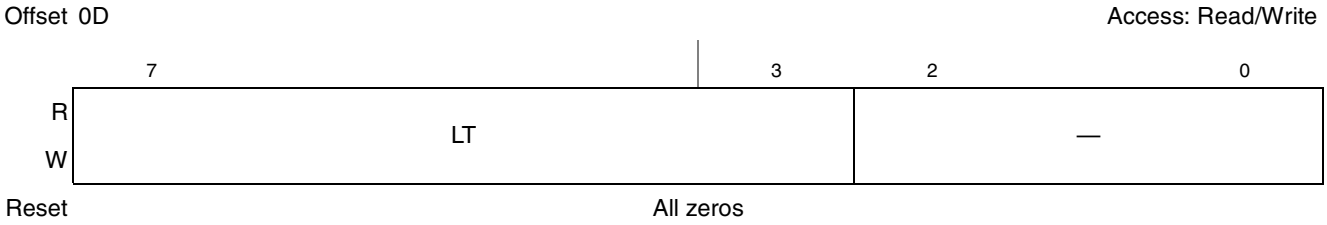


Figure 13-29. Latency Timer Configuration Register

Table 13-33 shows the bit settings of the latency timer register.

Table 13-33. Latency Timer Configuration Register Field Descriptions

Bits	Name	Description
7–3	LT	Latency timer. Specifies a granularity of 8 PCI clocks, the length of time that the PCI controller, when mastering a transaction, may hold the bus as the result of a bus grant. Refer to the PCI 2.3 specification for the rules by which the PCI controller completes transactions when the timer has expired.
2–0	—	Reserved

13.3.3.11 Header Type Configuration Register

Figure 13-30 shows the read-only header type register, which is hard-wired to 0x00.

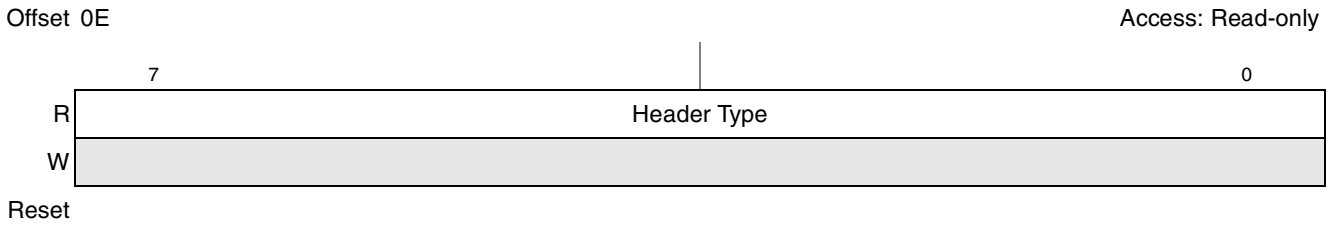


Figure 13-30. Header Type Configuration Register

13.3.3.12 BIST Control Configuration Register

Figure 13-31 shows the read-only BIST control register, which is hard-wired to 0x00.

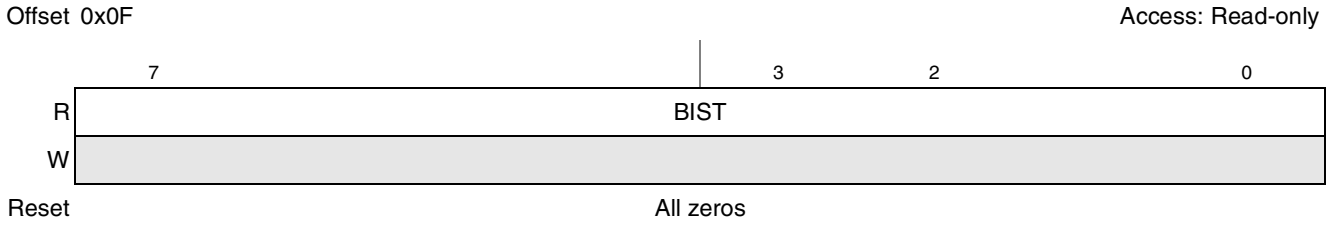


Figure 13-31. BIST Control Configuration Register

13.3.3.13 PIMMR Base Address Configuration Register

Figure 13-32 shows the PIMMR base address register fields.

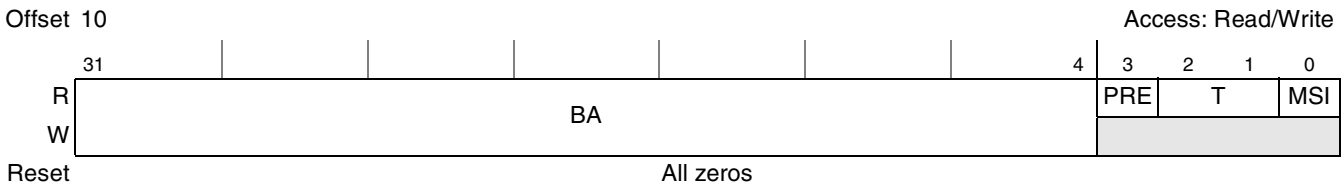


Figure 13-32. PIMMR Base Address Configuration Register

Table 13-34 shows the bit settings of the PIMMR base address register.

Table 13-34. PIMMR Base Address Configuration Register Field Descriptions

Bits	Name	Description
31–4	BA	Base address. Defines the base address for the internal (on-chip) memory-mapped register space. The size of this space is 1 MB.
3	PRE	Prefetchable. Hard-wired to 0.
2–1	T	Type. Hard-wired to 00.
0	MSI	Memory space indicator. Hard-wired to 0

13.3.3.14 GPL Base Address Register 0

The GPL base address register 0 is provided to allow access to local memory space. This register is closely tied to PIBAR0 and PIWAR0 in the CSR memory space. A write to GPL base address register 0 also causes a change in the base address bits that are not masked according to the IWS field of PIWAR0 in PIBAR0. Note that this write operation will not change the bits that are masked by the IWS field. For read operation these masked bits will always return zeros.

Figure 13-33 shows the GPL base address register 0 fields.

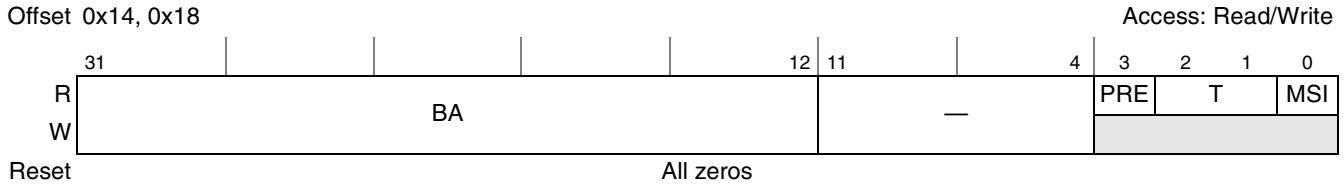


Figure 13-33. GPL Base Address Register 0

Table 13-35 shows the bit settings of the GPL base address register 0.

Table 13-35. GPL Base Address Register 0 Field Descriptions

Bits	Name	Description
31–12	BA	Base address. Defines the base address for the inbound window. Bits 11–4 are hard-wired to 0 since the minimum window size is 4 Kbytes.
3	PRE	Prefetchable. This bit is read-only and contains the value of the PF bit in PIWAR0.
2–1	T	Type. Hard-wired to 00.
0	MSI	Memory space indicator. Hard-wired to 0

13.3.3.15 GPL Base Address Registers 1–2

The general purpose local access base address registers are provided to allow access to local memory space. These registers are closely tied to PIBAR*n* and PIWAR*n* in the CSR memory space. A write to a GPL base address register also causes a change in the base address bits that are not masked according to the IWS field of PIWAR*n* in the corresponding PIBAR*n*. Note that this write operation will not change the bits that are masked by the IWS field. For read operations, these masked bits always return zeros. Figure 13-34 shows the GPL base address register 1–2 fields.

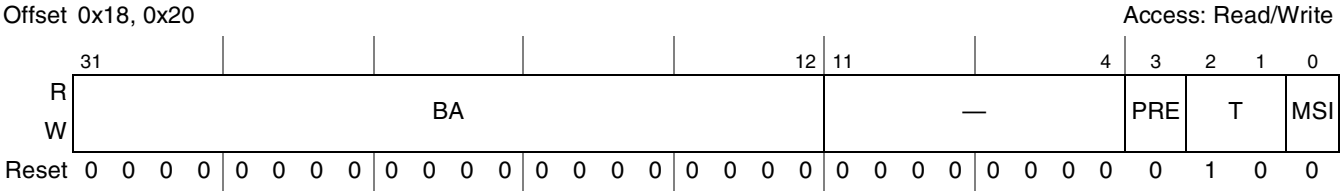


Figure 13-34. GPL Base Address Registers 1–2

Table 13-36 shows the bit settings of the GPL base address register 1–2.

Table 13-36. GPL Base Address Register 1,2 Field Descriptions

Bits	Name	Description
31–12	BA	Base address. Defines the two portion of the base address for the inbound window. Bits 11–4 are hard-wired to 0 since the minimum window size is 4 Kbytes.
11–4	—	Reserved
3	PRE	Prefetchable. This bit is read-only and contains the value of the PF bit in PIWAR*n*.
2–1	T	Type. Hard-wired to 10.
0	MSI	Memory space indicator. Hard-wired to 0

13.3.3.16 GPL Extended Base Address Registers 1–2

Two general-purpose local access base address registers are provided to allow access to local memory space. These registers are closely tied to PIBAR*n*, PIEBAR*n*, and PIWAR*n* in the CSR memory space. A write to a GPL extended base address register also causes a change in the base address bits that are not masked according to the IWS field of PIWARx in the corresponding PIBAR*n*/PIEBAR*n*. Note that this write operation does not change bits that are masked by the IWS field. For read operations these masked bits will always return zeros.

Figure 13-35 shows the GPL extended base address registers 1–2 fields.

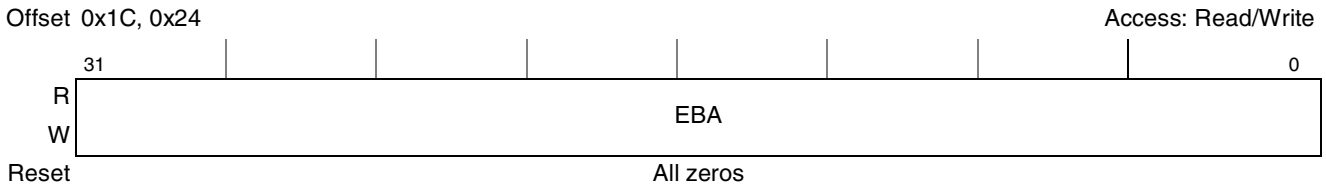


Figure 13-35. GPL Extended Base Address Registers 1–2

Table 13-37 shows the bit settings of the GPL extended base address register 1–2.

Table 13-37. GPL Extended Base Address Registers 1–2 Field Descriptions

Bits	Name	Description
31–0	EBA	Extended base address. Defines the high portion of the base address for the inbound window.

13.3.3.17 Subsystem Vendor ID Configuration Register

Figure 13-36 shows the subsystem vendor ID fields. The subsystem vendor ID configuration register is read-only from the PCI bus, but it can be programmed from the CSB.

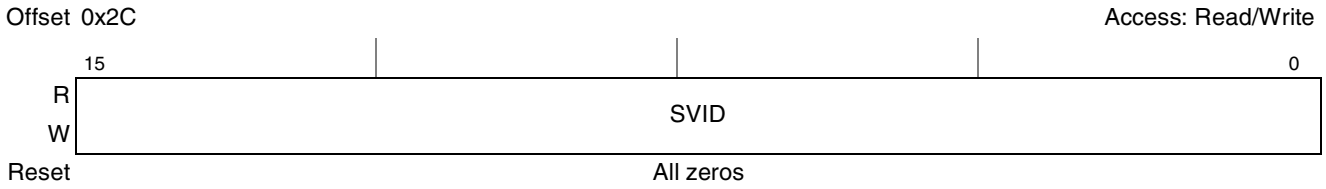


Figure 13-36. Subsystem Vendor ID Configuration Register

Table 13-38 shows the bit settings of the subsystem vendor ID configuration register.

Table 13-38. Subsystem Vendor ID Configuration Register Field Descriptions

Bits	Name	Description
15–0	SVID	Subsystem vendor ID. Identifies the manufacturer of the board or subsystem that contains this device.

13.3.3.18 Subsystem Device ID Configuration Register

Figure 13-37 shows the subsystem device configuration register ID fields. The subsystem device ID configuration register is read-only from the PCI bus, but it can be programmed from the CSB.

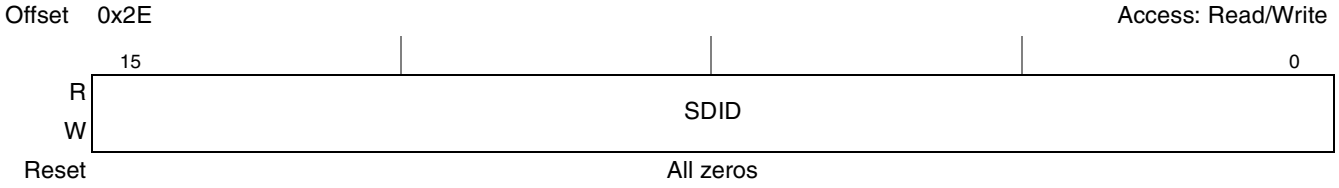


Figure 13-37. Subsystem Device ID Configuration Register

Table 13-39 shows the bit settings of the subsystem device ID configuration register.

Table 13-39. Subsystem Device ID Configuration Register Field Descriptions

Bits	Name	Description
15–0	SDID	Subsystem device ID. This field identifies the board or subsystem that contains this device.

13.3.3.19 Capabilities Pointer Configuration Register

The capabilities pointer register specifies the byte offset in the PCI configuration space that contains the first item in the capabilities list. Figure 13-38 shows the capabilities pointer configuration register fields.

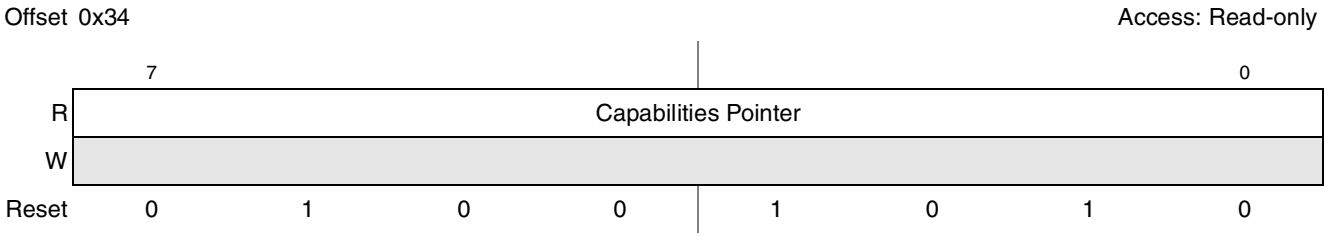


Figure 13-38. Capabilities Pointer Configuration Register

13.3.3.20 Interrupt Line Configuration Register

Figure 13-39 shows the interrupt line configuration register fields.

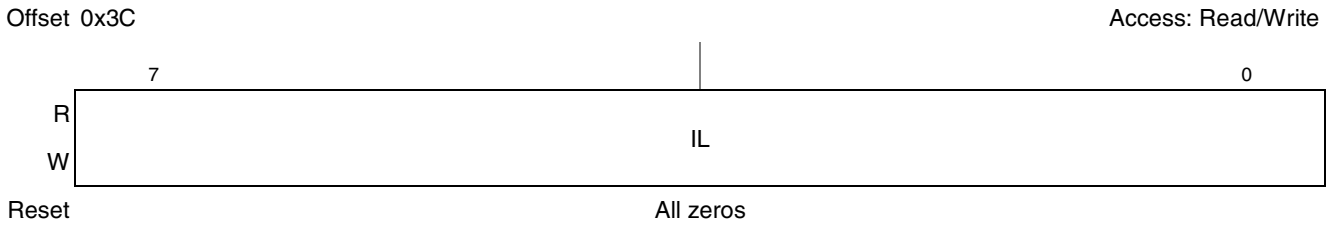


Figure 13-39. Interrupt Line Configuration Register

Table 13-40 shows the bit settings of the interrupt line configuration register.

Table 13-40. Interrupt Line Configuration Register Field Descriptions

Bits	Name	Description
7–0	IL	Interrupt line. Used to communicate interrupt line routing information. The value has no effect on the operation of the PCI controller.

13.3.3.21 Interrupt Pin Configuration Register

The interrupt pin configuration register tells which interrupt pin is used (0x01 means INTA). Figure 13-40 shows the interrupt pin configuration register fields.

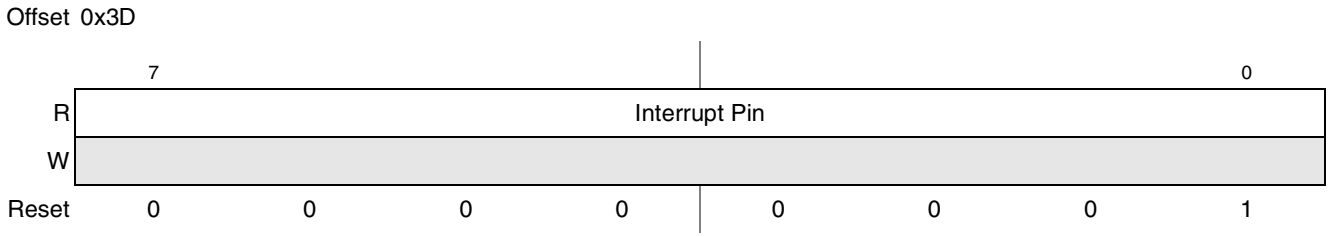


Figure 13-40. Interrupt Pin Register

13.3.3.22 Minimum Grant Configuration Register

Figure 13-41 shows the minimum grant configuration register fields.

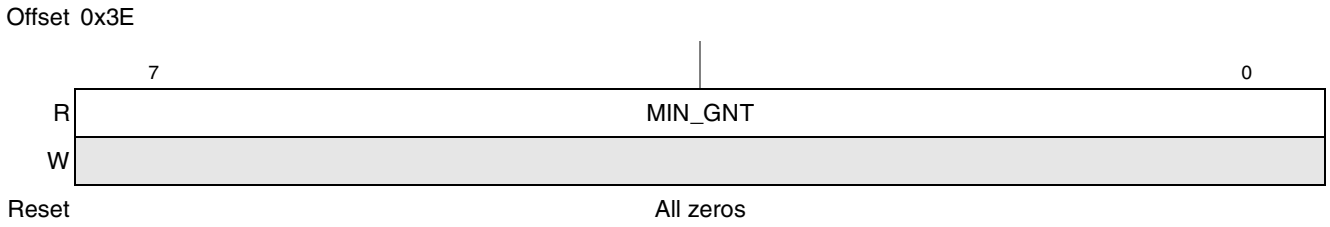


Figure 13-41. Minimum Grant Configuration Register

13.3.3.23 Maximum Latency Configuration Register

Figure 13-42 shows the maximum latency configuration register fields.

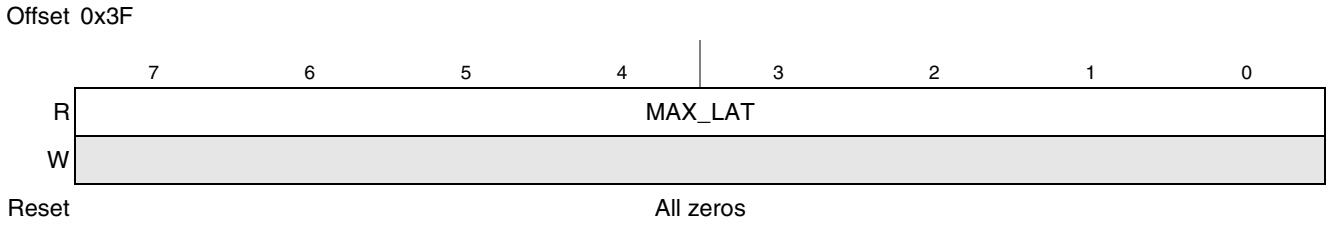
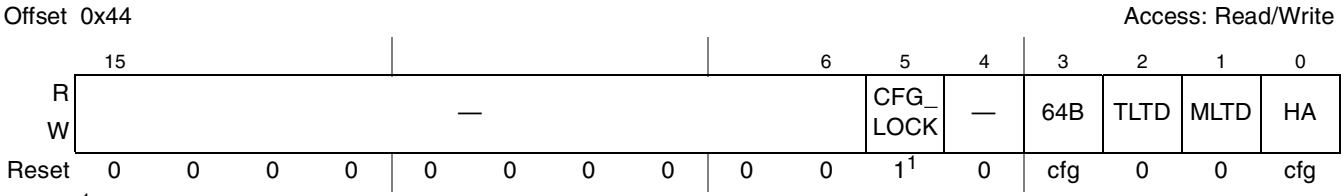


Figure 13-42. Maximum Latency Configuration Register

13.3.3.24 PCI Function Configuration Register

Figure 13-43 shows the PCI function configuration register fields.



1 When using a hard-coded reset configuration option, the CFG_LOCK bit is cleared at reset to enable configuration space accesses,

Figure 13-43. PCI Function Configuration Register

Table 13-41 shows the bit settings of the PCI function configuration register.

Table 13-41. PCI Function Configuration Register Field Descriptions

Bits	Name	Description
15–6	—	Reserved
5	CFG_LOCK	Configuration lock. Controls access to the PCI configuration space from the PCI port. In host mode the PCI configuration space is always inaccessible, so this bit is not used. Normally, this bit will be cleared in agent mode once the configuration of the PCI controller is complete to allow an external host to access the PCI configuration space. 0 Access to the configuration spaces is permitted. 1 Any inbound PCI access to the PCI configuration space is retried. See Section 4.3.1.1, “Reset Configuration Word Source,” on page 4-9 for more information on reset configuration.
4	—	Reserved
3	64B	64-bit data support. Provides the value of the PCI_64 — PCI 64-bit configuration bit is sampled at the end of the reset sequence. See Chapter 4, “Reset, Clocking, and Initialization,” for more information on the reset configuration words.
2	TLTD	Target latency timeout disable. Determines whether the PCI controller, while acting as a PCI target, times out when the first data phase of a transaction has not completed in 16 PCI cycles. 0 Target latency timeout enabled. 1 Target latency timeout disabled.

Table 13-41. PCI Function Configuration Register Field Descriptions (continued)

Bits	Name	Description
1	MLTD	Master latency timer disable. Determines whether the PCI controller, while acting as a PCI master, terminates a transaction upon the expiration of the master latency timer. 0 Master latency timer enabled. 1 Master latency timer disabled.
0	HA	Host/Agent. Indicates whether the PCI controller is in host mode or agent mode. It provides the value of the PCI_HOST—PCI host configuration bit is sampled at the end of the reset sequence. 0 Host mode 1 Agent mode

13.3.3.25 PCI Arbiter Control Register (PCIACR)

Figure 13-44 shows the PCI arbiter control register (PCIACR) fields.

Offset 0x46 Access: Read/Write

	15	14	13	12	11	7	6				2	1	0
R W	AD	PM	—	PMBD	—		PRI0	PRI1	PRI2	PRI3	PRI4	—	MPRI
Reset	cfg	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-44. PCI Arbiter Control Register (PCIACR)

Table 13-42 shows the bit settings of the PCIACR.

Table 13-42. PCI Arbiter Control Register (PCIACR) Field Descriptions

Bits	Name	Description
15	AD	Arbiter disable. Indicates whether the PCI controller functions as the arbiter for the PCI bus. It provides the value of the PCI arbiter enable configuration bit as sampled at the end of the reset sequence. See Chapter 4, “Reset, Clocking, and Initialization,” for more information on reset configuration. 0 Arbiter enabled 1 Arbiter disabled
14	PM	Parking mode. Controls which device receives a bus grant when there are no outstanding bus requests and the bus is idle. 0 The bus is parked with the last device to use the bus. 1 The bus is parked with the PCI controller.
13	—	Reserved
12	PBMD	PCI broken master disable. Determines whether the PCI controller ignores the bus requests of an initiator that requests the bus for an excessive period without using it. 0 An initiator that requests the bus and receives the grant must begin using the bus within 16 PCI clock periods after the bus becomes idle or its request is subsequently ignored. 1 No requests are ignored.
11–7	—	Reserved

Table 13-42. PCI Arbiter Control Register (PCIACR) Field Descriptions (continued)

Bits	Name	Description
6–2	PRI*n*	Priority level for master *n*. When the PCI controller functions as the arbiter for the PCI bus, each PRIn bit determines the arbitration priority level for the PCI master connected to the REQ*n*/GNT*n* pair. 0 Low priority 1 High priority
1	—	Reserved
0	MPRI	My priority. When the PCI controller functions as the arbiter for the PCI bus, this bit determines the arbitration priority level for the PCI controller when it acts as a PCI master. 0 Low priority 1 High priority

13.3.3.26 Hot Swap Register Block

Figure 13-45 shows the hot swap register block fields.

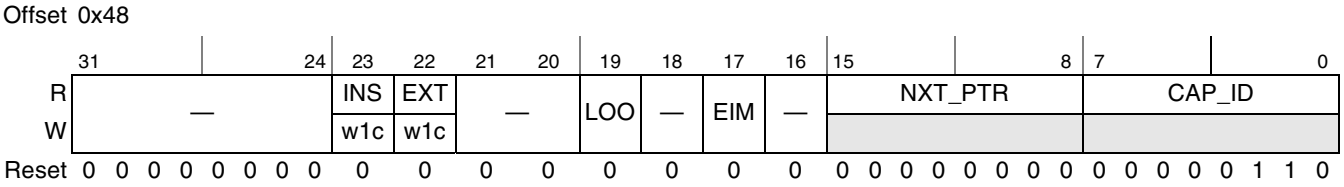


Figure 13-45. Hot Swap Register Block

Table 13-43 shows the bit settings of the Hot Swap register block.

Table 13-43. Hot Swap Register Block Field Descriptions

Bits	Name	Description
31–24	—	Reserved
23	INS	Insertion status. Indicates that a card has been inserted. Write 1 to clear this bit.
22	EXT	Extraction status. Indicates that a card has been extracted. Write 1 to clear this bit.
21–20	—	Reserved
19	LOO	LED On/Off. Controls the LED when the hardware is in state H2 0 LED off 1 LED on
18	—	Reserved
17	EIM	ENUM mask. This bit masks the CPCI_HS_ENUM input. 0 Enabled 1 Masked
16	—	Reserved
15–8	NXT_PTR	Next pointer—hardwired to 0x00 to indicate that this is the last item in the capabilities list.
7–0	CAP_ID	Capability ID for hot swap (hardwired to 0x06)

13.4 Functional Description

The following sections discuss the operation of the PCI controller.

13.4.1 PCI Bus Arbitration

The PCI bus arbitration approach is access-based. Bus masters must arbitrate for each access performed on the bus. PCI uses a central arbitration scheme where each master has its own unique request ($\overline{\text{REQ}}n$) output and grant ($\overline{\text{GNT}}n$) input signal. A simple request-grant handshake is used to gain access to the bus. Arbitration for the bus occurs during the previous access so that no PCI bus cycles are consumed waiting for arbitration (except when the bus is idle).

PCI 1 supports up to five external masters and PCI 2 supports up to three external masters. The two PCI buses together support a total of eight external masters, (besides the PCI controller itself) by using the $\overline{\text{REQ}}$ signals and generating the $\overline{\text{GNT}}$ signals.

During reset, the PCI controller samples the reset configuration bit (and programs the PCI_ARB_DIS bit accordingly) to determine if the arbiter is enabled or disabled. See Chapter 4, “Reset, Clocking, and Initialization,” for more information. The arbiter can also be enabled or disabled by directly programming the PCI_ARB_DIS bit in the arbiter configuration register (see Section 13.3.3.25, “PCI Arbiter Control Register (PCIACR),” for more information). However, it is recommended to use the reset configuration bit to set the arbiter state because the arbiter state controls the direction of $\overline{\text{REQ0}}$ and $\overline{\text{GNT0}}$.

If the arbiter is disabled, the PCI controller uses $\overline{\text{REQ0}}$ to issue requests to an external arbiter, and uses $\overline{\text{GNT0}}$ to receive grants from the external arbiter.

13.4.1.1 Bus Parking

When no devices are requesting the bus, the bus is granted, or parked, for a specified device to prevent the AD, PCI_C/$\overline{\text{BE}}$ and PCI_PAR signals from floating. The PCI controller can be configured to either park on itself or park on the last master to use the bus (see Section 13.3.3.25, “PCI Arbiter Control Register (PCIACR),” for more information).

13.4.1.2 Arbitration Algorithm

The round-robin arbitration algorithm has two priority levels. Each of the external PCI bus masters, plus the PCI controller, are assigned either a high or a low priority level, as programmed in the arbiter configuration register (see Section 13.3.3.25, “PCI Arbiter Control Register (PCIACR).”) Within each priority group (high or low), the bus grant is given to the next requesting device in numerical order, with the PCI controller itself positioned before device 0. $\overline{\text{GNT}}n$ is asserted for device *n* as soon as the previously granted device begins a transaction. Conceptually, the lowest priority device at any given time is the current bus master and the highest priority device is the next one to follow the current master. This is considered to be a fair algorithm because a given device cannot prevent other devices from having access to the bus—a given device automatically becomes the lowest priority device as soon as it begins to use the bus. If a master is not requesting the bus, the transaction slot is given to the next requesting device within the priority group.

The grant given to one device may be taken away and whenever a higher priority device asserts its request. If the bus is idle when a new device is to receive a grant, no device receives a grant for one clock; in the next clock, the new winner of the arbitration receives a grant. This operation allows for a turnaround clock when a device is using address stepping or when the bus is parked.

The low priority group collectively receives one bus transaction request slot in the high priority group. Therefore, if there are N high-priority devices, each high-priority device is guaranteed to get at least one of (N+1) bus transactions, and the M low priority devices are guaranteed to each get at least one of (N+1) x M bus transactions, with one of the low-priority devices receiving the grant in one of (N+1) bus transactions. If all devices are programmed to the same priority level or if there is only one device at the low priority, the algorithm provides each device an equal number of bus grants in a round-robin sequence.

An arbitration example with three masters in the high priority group and two in the low priority group is shown in Figure 13-46. Noting that one position in the high priority group is actually a place-holder for the low priority group, it can be seen that each high priority initiator is guaranteed at least 1 out of 3 transaction slots, and each low priority initiator is guaranteed at least 1 out of 6 slots. Assuming all devices are requesting the bus, the grant sequence (with device 1 being the current master) is as follows: 0, 2, the PCI controller, 0, 2, 1, 0, 2, the PCI controller, and so on. If, for example, device 2 is not requesting the bus, the grant sequence becomes 0, the PCI controller, 0, 1, 0, the PCI controller, and so on. If device 2 now requests the bus at a point in the sequence when device 0 is conducting a transaction and the PCI controller is the next grant, then the PCI controller's grant is removed, and the higher-priority device 2 is awarded the next grant.

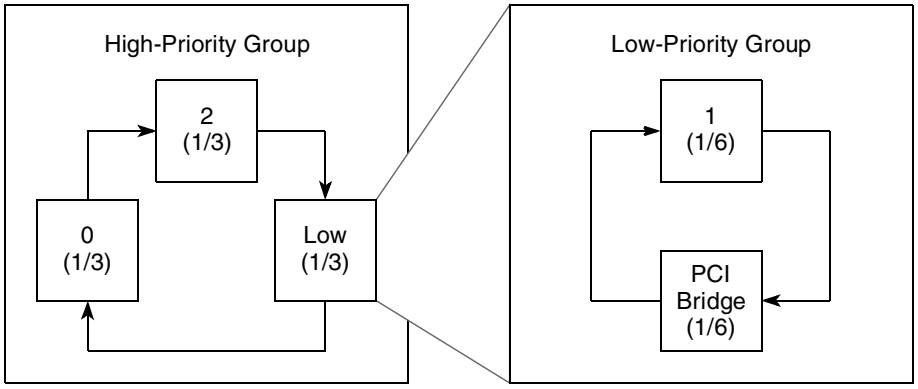


Figure 13-46. PCI Arbitration Example

13.4.1.3 Broken Master Lock-Out

The broken master feature allows the arbiter to lock out any masters that are broken or ill-behaved. This feature is controlled by programming the PCI arbiter control register. When the broken master feature is enabled, a granted device that does not assert $\overline{\text{PCI_FRAME}}$ within 16 PCI clock cycles after the bus is idle, has its grant removed and subsequent requests are ignored until its $\overline{\text{REQ}}$ is negated for at least one clock cycle. This prevents ill-behaved masters from monopolizing the bus. When the broken master feature is disabled, a device that requests the bus and receives a grant never loses its grant until and unless it begins a transaction or negates its $\overline{\text{REQ}}$ signal. Note that disabling the broken master feature is not recommended.

13.4.1.4 Master Latency Timer

The PCI controller implements the master latency timer register (see Section 13.3.3.10, "Latency Timer Configuration Register") to prevent itself from monopolizing the bus. When the master latency timer expires, the PCI controller checks the state of its PCI*n*_GNT signals. If the PCI*n*_GNT signal is not asserted, the PCI controller completes one more data phase and relinquishes the bus. The master latency timer can be disabled if needed (see Section 13.3.3.24, "PCI Function Configuration Register," for more information).

13.4.2 Bus Commands

PCI bus commands indicate the type of transaction occurring on the bus. These commands are encoded on PCI_C/BE[3:0] during the address phase of the transaction. PCI bus commands are described in Table 13-44.

Table 13-44. PCI Command Definitions

PCI_C/BE[3:0]	Command Type	Supported as:		Definition
		Initiator	Target	
0b0000	Interrupt acknowledge	Yes	No	A read implicitly addressed to the system interrupt controller. The size of the vector to be returned is indicated on the byte enables after the address phase.
0b0001	Special cycle	Yes	No	Provides a simple message broadcast mechanism. See Section 13.4.4.6, "Special Cycle Command," for more information.
0b0010	I/O read	Yes	No	Accesses agents mapped in I/O address space.
0b0011	I/O write	Yes	No	Accesses agents mapped in I/O address space.
0b010x	—	—	—	Reserved. No response occurs.
0b0110	Memory read	Yes	Yes	Accesses agents mapped in memory address space. A read from prefetchable space, when seen as a target, fetches a cache line of data (32 bytes) from the starting address, even though all 32 bytes may not actually be sent to the initiator.
0b0111	Memory write	Yes	Yes	Accesses agents mapped in memory address space.
0b100x	—	—	—	Reserved. No response occurs.
0b1010	Configuration read	Yes	Yes	Accesses the configuration space of each agent. An agent is selected when its IDSEL signal is asserted. See Section 13.4.4.4, "Host Mode Configuration Access," for more information on configuration accesses. As a target, a configuration read is only accepted if the PCI controller is configured to be in agent mode.
0b1011	Configuration write	Yes	Yes	Accesses the configuration space of each agent. An agent is selected when its IDSEL signal is asserted. See Section 13.4.4.4, "Host Mode Configuration Access," for more information. As a target, a configuration write is only accepted if the PCI controller is configured to be in agent mode.
0b1100	Memory read multiple	Yes	Yes	Causes a prefetch of the next cache line.
0b1101	Dual address cycle	No	Yes	Transfers an 8-byte address to devices.

Table 13-44. PCI Command Definitions (continued)

PCI_C/BE[3:0]	Command Type	Supported as:		Definition
		Initiator	Target	
0b1110	Memory read line	Yes	Yes	Indicates that the initiator intends to transfer an entire cache line of data.
0b1111	Memory write and invalidate	No	Yes	Indicates that the initiator will transfer an entire cache line of data, and if PCI has any cacheable memory, this line needs to be invalidated.

13.4.3 PCI Protocol Fundamentals

The bus transfer mechanism on the PCI bus is called a burst. A burst is comprised of an address phase and one or more data phases.

All signals are sampled on the rising edge of the PCI clock. Each signal has a setup and hold window with respect to the rising clock edge, in which transitions are not allowed. Outside this aperture, signal values or transitions have no significance.

13.4.3.1 Basic Transfer Control

PCI data transfers are controlled by the following signals:

- PCI_FRAME is driven by an initiator to indicate the beginning and end of a transaction.
- PCI_IRDY (initiator ready) is driven by an initiator, allowing it to force wait cycles.
- PCI_TRDY (target ready) is driven by a target, allowing it to force wait cycles.

The bus is idle when both PCI_FRAME and PCI_IRDY are negated. The first clock cycle in which PCI_FRAME is asserted indicates the beginning of the address phase. The address and the bus command code are transferred in that cycle. The next cycle ends the address phase and begins the data phase.

During the data phase, data is transferred in each cycle that both PCI_IRDY and PCI_TRDY are asserted. Once the PCI controller, as an initiator, has asserted PCI_IRDY, it does not change PCI_IRDY or PCI_FRAME until the current data phase completes, regardless of the state of PCI_TRDY. Once the PCI controller, as a target, has asserted PCI_TRDY or PCI_STOP it does not change PCI_DEVSEL, PCI_TRDY, or PCI_STOP until the current data phase completes.

When the PCI controller (as a master) intends to complete only one more data transfer, PCI_FRAME is negated and PCI_IRDY is asserted (or kept asserted) indicating the initiator is ready. After the target indicates it is ready (PCI_TRDY asserted) the bus returns to the idle state.

13.4.3.2 Addressing

The PCI specification defines three physical address spaces—memory, I/O, and configuration. The memory and I/O address spaces are standard for all systems. The configuration address space supports the PCI hardware configuration. Each PCI device decodes the address for each PCI transaction with each agent responsible for its own address decode.

The information contained in the two lower address bits (AD1 and AD0) depends on the address space. In the I/O address space, all 32 address/data lines provide the full byte address. AD[1:0] are used for the generation of PCI*n*_DEVSEL and indicate the least significant valid byte involved in the transfer. Once a target has claimed an I/O access, it first determines if it can complete the entire access as indicated by the byte enable signals. If all the selected bytes are not in the address range, the entire access should not be completed; that is, the target should not transfer any data and should terminate the transaction with a target-abort operation. See Section 13.4.3.6, "Bus Transactions," for more information.

In the configuration address space, accesses are decoded to a 4-byte address using AD[7:2]. An agent determines if it is the target of the access when a configuration command is decoded, IDSEL is asserted, and AD[1:0] are 0b00; otherwise, the agent ignores the current transaction. The PCI controller determines a configuration access is for a device on the PCI bus by decoding a configuration command. When in agent mode, the PCI controller responds to host-generated PCI configuration cycles when its IDSEL is asserted during a configuration cycle.

For memory accesses, the address is decoded using AD[31:2]; thereafter, the address is incremented internally by 4 bytes until the end of the burst transfer. Another initiator in a memory access should drive 0b00 on AD[1:0] during the address phase to indicate a linear incrementing burst order. The PCI controller checks AD[1:0] during a memory command access and provides the linear incrementing burst order. On reads, if AD[1:0] is 0b10, which represents a cache line wrap, the PCI controller linearly increments the burst order starting at the critical 64-bit address, wraps at the end of the cache line, and disconnects after reading one cache line. If AD[1:0] is 0bx1 (a reserved encoding) and the PCI_C/BE[3:0] signals indicate a memory transaction, it executes a target disconnect after the first data phase is completed. Note that AD[1:0] are included in parity calculations.

13.4.3.3 Device Selection

As a target, the PCI controller drives PCI_DEVSEL one clock following the address phase as indicated in the configuration space status register; see Section 13.3.3.4, "PCI Status Configuration Register," for more information. The PCI controller as a target qualifies the address/data lines with PCI_FRAME before asserting PCI_DEVSEL. The PCI_DEVSEL signal is asserted at or before the clock edge at which the PCI controller enables its PCI_TRDY, PCI_STOP, or data (for a read). The PCI_DEVSEL signal is not negated until PCI_FRAME is negated, with PCI_IRDY asserted and either PCI_STOP or PCI_TRDY asserted. The exception to this is a target-abort; see Section 13.4.3.8, "Transaction Termination," for more information.

As an initiator, if the PCI controller does not see the assertion of PCI_DEVSEL within 4 clocks of PCI_FRAME, it terminates the transaction with a master-abort as described in Section 13.4.3.8, "Transaction Termination," for more information.

13.4.3.4 Byte Enable Signals

The byte enable signals (BE[3:0]) indicate which byte lanes carry valid data. The byte enable signals may enable different bytes for each of the data phases. The byte enable signals are valid on the edge of the clock that starts each data phase and remain valid for the entire data phase.

If the PCI controller, as a target, sees no byte enable signals asserted, it completes the current data phase with no permanent change. This implies that on a read transaction, the PCI controller expects the data not to be changed, and on a write transaction, the data is not stored.

13.4.3.5 Bus Driving and Turnaround

The turnaround-cycle is one clock cycle and is required to avoid contention. This cycle occurs at different times for different signals. PCI_IRDY, PCI_TRDY, and PCI_DEVSEL use the address phase as their turnaround-cycle. PCI_FRAME, PCI_C/BE[3:0], and AD[31:0] use the idle cycle between transactions as their turnaround-cycle. (An idle cycle in PCI is when both PCI_FRAME and PCI_IRDY are negated).

Byte lanes not involved in the current data transfer are driven to a stable condition even though the data is not valid.

13.4.3.6 Bus Transactions

The timing diagrams in this section show the relationship of significant signals involved in bus transactions.

Note the following conventions:

- When a signal is drawn as a solid line, it is actively being driven by the current initiator or target.
- When a signal is drawn as a dashed line, no agent is actively driving it.
- Three-stated signals with slashes between the two rails have indeterminate values.
- The terms ‘edge’ and ‘clock edge’ refer to the rising edge of the clock.
- The terms ‘asserted’ and ‘negated’ refer to the globally visible state of the signal on the clock edge, and not to signal transitions.
- The symbol represents a turnaround-cycle.

13.4.3.7 Read and Write Transactions

Both read and write transactions begin with an address phase followed by a data phase. The address phase occurs when PCI*n*_FRAME is asserted for the first time, and the AD[31:0] signals contain a byte address and the PCI_C/BE[3:0] signals contain a bus command. The data phase consists of the actual data transfer and possible wait cycles; the byte enable signals remain actively driven from the first clock of the data phase through the end of the data transfer.

A read transaction starts when PCI*n*_FRAME is asserted for the first time and the PCI_C/BE[3:0] signals indicate a read command. Figure 13-47 shows an example of a single beat read transaction.

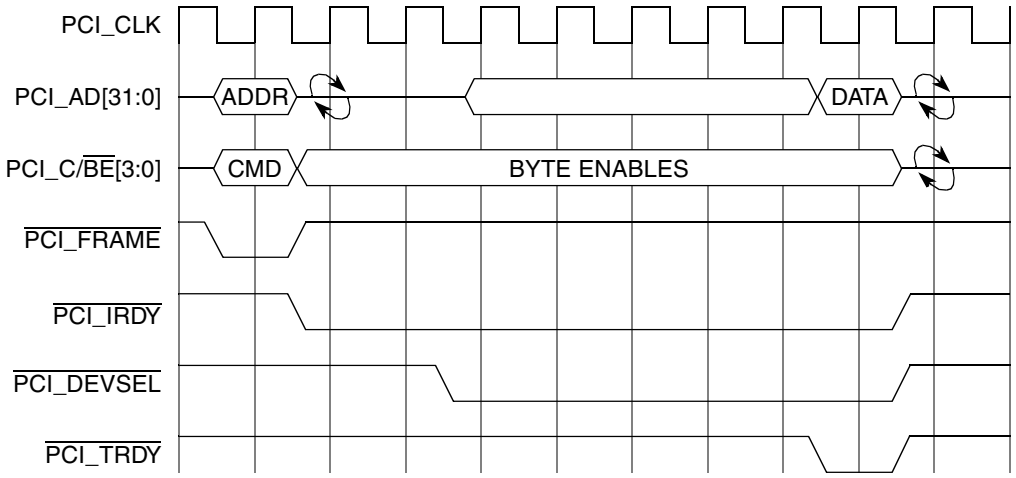


Figure 13-47. Single Beat Read Example

Figure 13-48 shows an example of a burst read transaction.

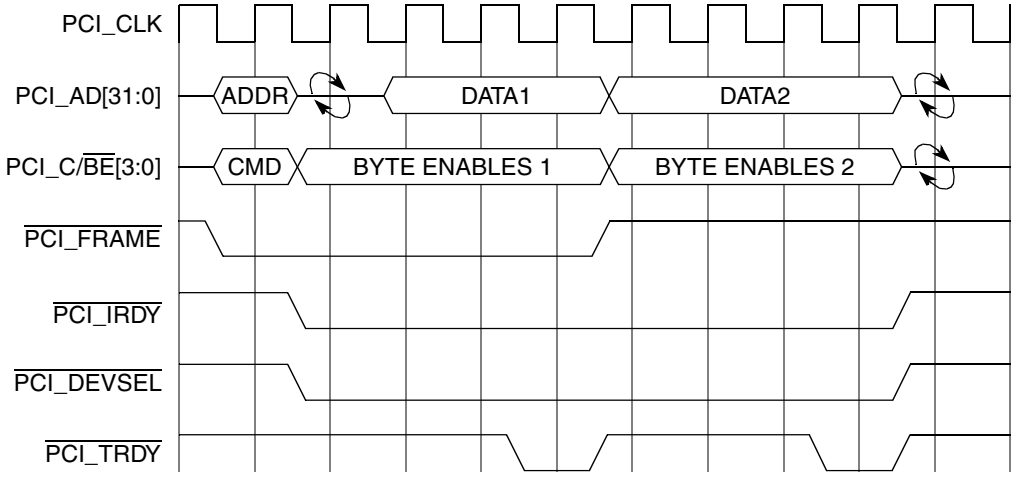


Figure 13-48. Burst Read Example

During the turnaround-cycle following the address phase, the PCI_C/BE[3:0] signals indicate which byte lanes are involved in the data phase. The turnaround-cycle must be enforced by the target with the PCI_TRDY signal if using fast PCI_DEVSEL assertion. The earliest the target can provide valid data is one cycle after the turnaround cycle. The target must drive the AD[31:0] signals when PCI_DEVSEL is asserted except during the turnaround cycle.

The data phase completes when data is transferred, which occurs when both PCI_IRDY and PCI_TRDY are asserted on the same clock edge. When either is negated, a wait cycle is inserted and no data is transferred. To indicate the last data phase PCI_IRDY must be asserted when PCI_FRAME is negated.

A write transaction starts when PCI_FRAME is asserted for the first time and the PCI_C/BE[3:0] signals indicate a write command. Figure 13-49 shows an example of a single-beat write transaction.

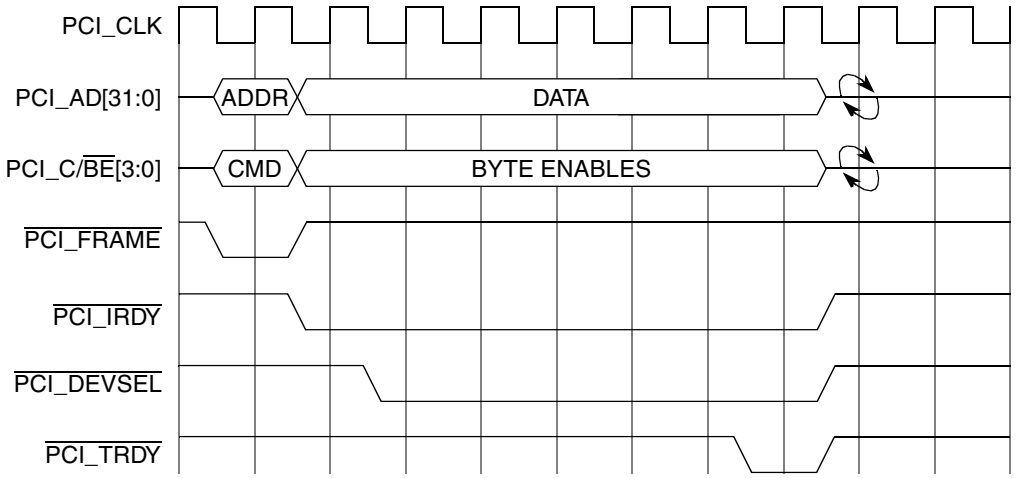


Figure 13-49. Single Beat Write Example

Figure 13-50 shows an example of a burst write transaction.

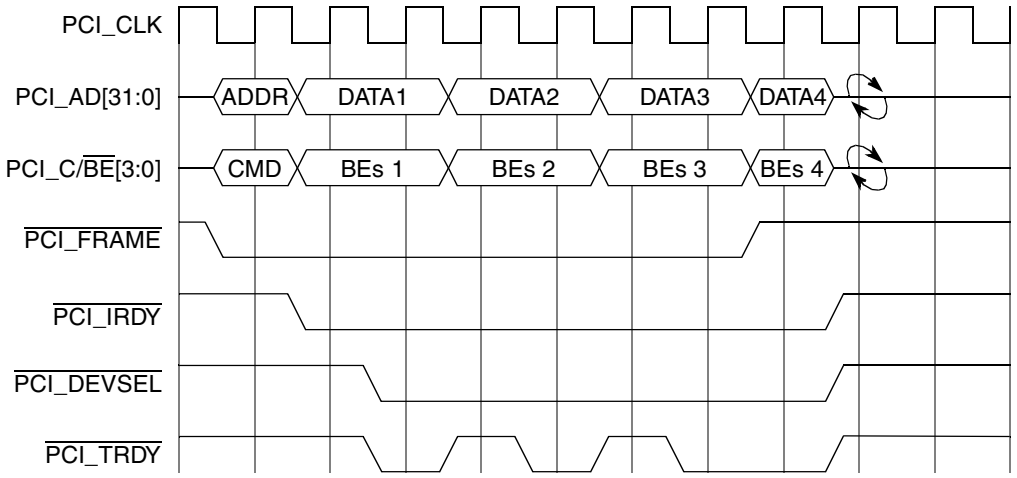


Figure 13-50. Burst Write Example

A write transaction is similar to a read transaction except no turnaround cycle is needed following the address phase because the initiator provides both address and data. Data phases are the same for both read and write transactions.

13.4.3.8 Transaction Termination

The termination of a PCI transaction is orderly and systematic, regardless of the cause of the termination. All transactions end when PCI*n*_FRAME and PCI*n*_IRDY are both negated, indicating the idle cycle.

The PCI controller as an initiator terminates a transaction when PCI*n*_FRAME is negated and PCI*n*_IRDY is asserted. This indicates that the final data phase is in progress. The final data transfer occurs when both PCI*n*_TRDY and PCI*n*_IRDY are asserted. A master-abort is an abnormal case of a master initiated

termination. If the PCI controller detects that PCI*n*_DEVSEL has remained negated for more than four clocks after the assertion of PCI_FRAME, it negates PCI*n*_FRAME and then, on the next clock, negates PCI*n*_IRDY. On aborted reads, the PCI controller returns 0xFFFF_FFFF. The data is lost on aborted writes.

When the PCI controller as a target needs to suspend a transaction, it asserts PCI*n*_STOP. Once asserted, PCI*n*_STOP remains asserted until PCI*n*_FRAME is negated. Depending on the circumstances, data may or may not be transferred during the request for termination. If PCI*n*_TRDY and PCI*n*_IRDY are asserted during the assertion of PCI*n*_STOP**,** data is transferred. This type of target-initiated termination is called a disconnect B, shown in Figure 13-51. If PCI*n*_TRDY is asserted when PCI*n*_STOP is asserted but PCI*n*_IRDY is not, PCI*n*_TRDY must remain asserted until PCI*n*_IRDY is asserted and the data is transferred. This is called a disconnect A target-initiated termination, also shown in Figure 13-51. However, if PCI*n*_TRDY is negated when PCI*n*_STOP is asserted, no more data is transferred, and the initiator therefore does not have to wait for a final data transfer (see the retry diagram in Figure 13-49).

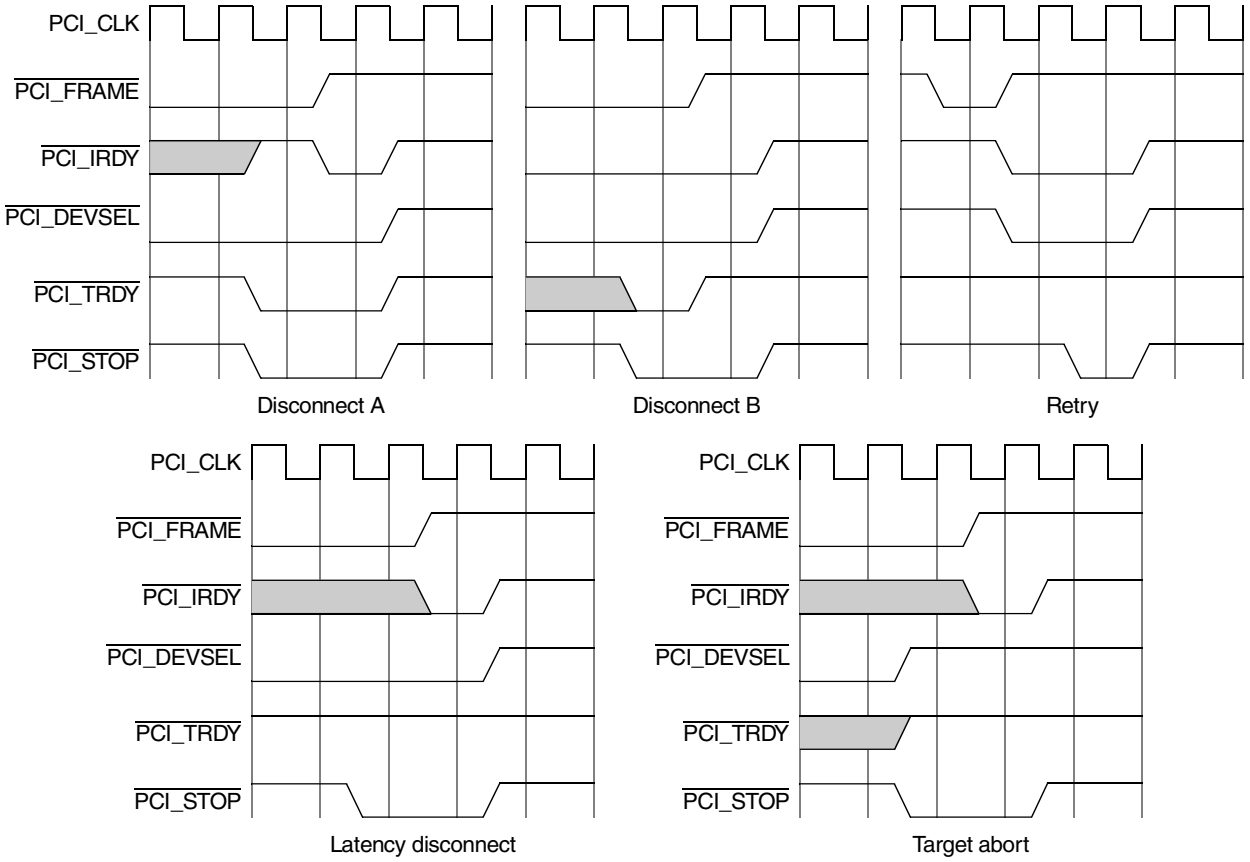


Figure 13-51. Target-Initiated Terminations

Note that when an initiator is terminated by PCI*n*_STOP, it must negate its REQ*n* signal for a minimum of two PCI clocks (of which one clock is needed for the bus to return to the idle state). If the initiator intends to complete the transaction, it should reassert its REQ*n* immediately following the two clocks or potential starvation may occur. If the initiator does not intend to complete the transaction, it can assert REQ*n* whenever it needs to use the PCI bus again.

The PCI controller terminates a transaction in the following cases:

- Eight PCI clock cycles have elapsed between data phases. This is a ‘latency disconnect’ (see Figure 13-49).
- AD[1:0] is 0bx1 **(**a reserved burst ordering encoding) during the address phase and one data phase has completed.
- The PCI command is a configuration command and one data phase has completed.
- A streaming transaction crosses a 4-Kbyte page boundary.
- A streaming transaction runs out of I/O sequencer buffer entries.
- A cache line wrap transaction has completed a cache line transfer.

Another target-initiated termination is the retry termination. Retry refers to termination requested because the target is currently in a state where it is unable to process the transaction. This can occur because no buffer entries are available in the I/O sequencer, or the sixteen clock latency timer has expired without transfer of the first data. The target latency timer of the PCI controller can be optionally disabled. See Section 13.3.3.24, “PCI Function Configuration Register,” for more information.

When the PCI controller is in host mode it does not respond to any PCI configuration transactions. When the PCI controller is in agent mode and the CFG_LOCK lock bit is set (see Section 13.3.3.24, “PCI Function Configuration Register”) the PCI controller retries all transactions to the PCI configuration space or the internal (on-chip) memory-mapped register space. Note that all retried accesses need to be completed. An example of a retry is shown in Figure 13-49.

Note that because a target can determine whether or not data is transferred (when both $\overline{\text{PCI}n\text{_IRDY}}$ and $\overline{\text{PCI}n\text{_TRDY}}$ are asserted), if it wants to do only one more data transfer and then stop, it may assert $\overline{\text{PCI}n\text{_TRDY}}$ and $\overline{\text{PCI}n\text{_STOP}}$ at the same time.

Target-abort refers to the abnormal termination that is used when a fatal error has occurred, or when a target will never be able to respond. Target-abort is indicated when $\overline{\text{PCI}n\text{_STOP}}$ is asserted and $\overline{\text{PCI}n\text{_DEVSEL}}$ is negated. This indicates that the target requires the transaction to be terminated and does not want the transaction tried again. Note that any transferred data may have been corrupted.

The PCI controller terminates a transaction with target-abort in the case in which it is the intended target of a read transaction from system memory and the data from memory is corrupt. If the PCI controller is the intended target of a transaction and an address parity error occurs, or a data parity error occurs on a write transaction to system memory, it continues the transaction on the PCI bus but aborts internally. The PCI controller does not target-abort in this case.

If the PCI controller is mastering a transaction that terminates with a target-abort, undefined data is returned on a read and write data is lost. An example of a target-abort is shown in Figure 13-49.

An initiator may retry any target disconnect accesses, except target-abort, at a later time starting with the address of the next non-transferred data. Retry is actually a special case of disconnect where no data transfer occurs at all and the initiator must start the entire transaction over again.

13.4.4 Other Bus Operations

The following sections provide information on additional PCI bus operations.

13.4.4.1 Fast Back-to-Back Transactions

In the two types of fast back-to-back transactions, the first type places the burden of avoiding contention on the initiator while the second places the burden on all potential targets. The PCI controller as a target supports both types of fast back-to-back transactions but does not support them as an initiator. The PCI controller as a target has the fast back-to-back enable bit hardwired to one, that is, enabled.

For the first type (governed by the initiator), the initiator may only run a fast back-to-back transaction to the same target. For the second type, when the PCI controller detects a fast-back-to-back operation and did not drive $\overline{\text{PCI}n\text{_DEVSEL}}$ in the previous cycle, it delays the assertion of $\overline{\text{PCI}n\text{_DEVSEL}}$ and $\overline{\text{PCI}n\text{_TRDY}}$ for one cycle to allow the other target to get off the bus.

13.4.4.2 Dual Address Cycles

The PCI controller supports dual address cycle (DAC) commands (64-bit addressing on PCI bus) as a target. DACs are different from single address cycles (SACs) in that the address phase takes two PCI beats instead of one PCI beat to transfer (64-bit vs. 32-bit addressing). Only PCI memory commands can use DAC cycles; I/O, configuration, interrupt acknowledge, and special cycle command cannot use DAC cycles. The PCI controller supports single-beat and burst DAC transactions.

13.4.4.3 Data Streaming

The PCI controller provides data streaming for PCI transactions to and from prefetchable memory. In other words, when the PCI controller is a target for a PCI initiated transaction, it supplies or accepts multiple cache lines of data without disconnecting. For PCI transactions to non-prefetchable space, the PCI controller disconnects after the first data phase so streaming cannot occur.

For PCI memory reads, streaming is achieved by performing speculative reads from memory in prefetchable space. A block of memory may be marked as prefetchable by setting the PCI configuration registers bit for the inbound address translation (see Section 13.3.2.15, “PCI Inbound Window Attribute Registers (PIWARn),” for more information) in the following cases:

- When reads do not alter the contents of memory (reads have no side effects)
- When reads return all bytes regardless of the byte enable signals
- When writes can be merged without causing errors

For a memory read command or a memory read line command, the PCI controller reads one cache line from memory. If the transaction crosses a cache line boundary, the PCI controller starts the read of a new cache line. For a memory read multiple command, the PCI controller reads two cache lines from memory. When the PCI transaction finishes the read for the first cache line, the PCI controller performs a speculative read of a third cache line. The PCI controller continues this prefetching until the end of the transaction.

For PCI writes to memory, streaming is achieved by buffering the transaction in the space available within the I/O sequencer. This allows PCI memory writes to execute with no wait states.

A disconnect occurs if the PCI controller runs out of buffer space on writes, or the PCI controller cannot supply consecutive data beats for reads within eight PCI bus clocks of each other. A disconnect also occurs if the transaction crosses a 4-Kbyte page boundary.

13.4.4.4 Host Mode Configuration Access

The PCI controller provides two types of configuration accesses to support hierarchical bridges. To access configuration space, a value is written to the CONFIG_ADDR register specifying which PCI bus, which device, and which configuration register to be accessed.

When the PCI controller sees an access that falls inside the 4 bytes beginning at the CONFIG_DATA address, it checks the enable bit, the device number and the bus number in the CONFIG_ADDR register. If the enable bit is set and the device number is not equal to all ones, a configuration cycle translation is performed. When the device number field is equal to all ones, it has a special meaning (see Section 13.4.4.6, “Special Cycle Command,” for more information).

There are two types of translations supported:

- Type 0 translations—For when the device is on the PCI bus connected to the PCI controller.
- Type 1 translations—For when the device is on another bus somewhere behind the PCI controller.

For type 0 translations, the PCI controller decodes the device number field to assert the appropriate IDSEL line and perform a configuration cycle on the PCI bus with AD[1:0] as 0b00. All 21 IDSEL bits are decoded, starting with bit AD11. That is, if the device number field contains 0b01011, AD11 on the PCI bus is set. The IDSEL lines are bit-wise associated with increasing values for the device number such that AD12 corresponds to 0b01100, and so on up to bit 30 as shown in Table 13-41. AD31 is selected with 0b01010. A device number of 0b11111 indicates a special cycle. Device number 0b00000 is used for configuring the PCI controller itself. Bits 10 through 8 are copied to the PCI bus as an encoded value for components which contain multiple functions. Bits 7 through 2 are also copied onto the PCI bus. The PCI controller implements address stepping on configuration cycles so that the target’s PCI*n*_IDSEL, which is connected directly to one of the AD lines, reaches a stable value. This means that a valid address and command are driven on the AD and PCI_C/$\overline{\text{BE}}$ lines one cycle before the assertion of $\overline{\text{PCI}n\text{_FRAME}}$.

For type 1 translations, the PCI controller copies the contents of the CONFIG_ADDR register directly onto the PCI address/data lines during the address phase of a configuration cycle, with the exception that AD[1-0] contains 0b01 (not 0b00 as in Type 0 translations).

When the PCI controller is configured as a host device, a local master sometimes needs to perform configuration reads from unpopulated PCI slots (as part of the system configuration). To avoid getting a machine check interrupt, the following steps should be taken:

1. Mask the NORSP bit in the error mask register. See Section 13.3.2.10, “PCI Error Control Register (PCI_ECR).”
2. Perform the PCI configuration reads.
3. Clear the NORSP bit in the error status register.
4. Unmask (write 1) the NORSP bit in the error mask register. See Section 13.3.2.3, “PCI Error Enable Register (PCI_EER).”

13.4.4.5 Agent Mode Configuration Access

When the PCI controller is configured as an agent device, it responds to remote host generated PCI configuration accesses to the PCI interface. This is indicated by decoding the configuration command along with the PCI controller's IDSEL being asserted. A remote host can access the 256-byte PCI configuration area and the memory-mapped configuration registers within the PCI controller.

13.4.4.6 Special Cycle Command

A special cycle command contains no explicit destination address but is broadcast to all PCI agents. Each receiving agent must determine whether the message is applicable to itself. No assertion of $\overline{\text{PCI}n\text{_DEVSEL}}$ in response to a special cycle command is necessary.

A special cycle command is like any other bus command in that it has an address phase and a data phase. The address phase starts like all other commands with the assertion of $\overline{\text{PCI}n\text{_FRAME}}$ and completes when $\overline{\text{PCI}n\text{_FRAME}}$ and $\overline{\text{PCI}n\text{_IRDY}}$ are negated. Special cycles terminate with a master-abort. (In the special cycle case, the received-master-abort bit in the configuration status register is not set.)

The address phase contains no valid information other than the command field. Even though there is no explicit address, the address/data lines are driven to a stable state and parity is generated. During the data phase, the address/data lines contain the message type and an optional data field. The message is encoded on the sixteen least-significant bits (AD[15:0]). The data field is encoded on AD[31:16]. When running a special cycle, the message and data are valid on the first clock $\overline{\text{PCI}n\text{_IRDY}}$ is asserted.

When the PCI_CONFIG_ADDRESS register is written with a value so that the bus number matches the bridge bus, the device number is all ones, the function number is all ones, and the register number is zero. The next time the PCI_CONFIG_DATA register is accessed, the PCI controller executes either a special cycle or an interrupt acknowledge command. When the PCI_CONFIG_DATA register is written, the PCI controller generates a special cycle encoding on the command/byte enable lines during the address phase and drives the data from the PCI_CONFIG_DATA register onto the address/data lines during the first data phase.

If the bus number field of the PCI_CONFIG_ADDRESS does not match one of the PCI controller bus numbers, the PCI controller passes the write to PCI_CONFIG_DATA through to the PCI bus as a type 1 configuration cycle as it does any other time the bus number field does not match.

Table 13-45. Special Cycle Commands

Address (AD[15–0])	Message Type	Description
0x0000	SHUTDOWN (SLEEP)	Indicates the processor is entering its most power saving mode
0x0001	HALT (DOZE)	Indicates the processor is entering a power save mode where address decoding is still available
0x0002–0xFFFF	—	Reserved for future commands

13.4.4.7 Interrupt Acknowledge

When the PCI_CONFIG_ADDRESS register is written with a value such that the bus number is 0x00, the device number is all ones, the function number is all ones, and the register number is zero, the next time the PCI_CONFIG_DATA register is accessed the PCI controller does either a special cycle command or an interrupt acknowledge command. When the PCI_CONFIG_DATA register is read, the PCI controller generates an interrupt acknowledge command encoding on the command/byte enable lines during the address phase. During the address phase, AD[31:0] do not contain a valid address but are driven with stable data and valid parity (PCI*n*_PAR). During the data phase, the byte enable signals determine which bytes are involved in the transaction. The interrupt vector must be returned when $\overline{\text{PCI}n\text{_TRDY}}$ is asserted.

An interrupt acknowledge transaction can also be issued on the PCI bus by reading from the PCI_INT_ACK register.

13.4.5 Error Functions

This section describes PCI bus errors.

13.4.5.1 Parity

During valid 32-bit address and data transfers, parity covers all 32 address/data lines and the 4 command/byte enable lines regardless of whether or not all lines carry meaningful information. Byte lanes not actually transferring data are driven with stable (albeit meaningless) data and are included in the parity calculation. During configuration, special cycle or interrupt acknowledge commands, some address lines are not defined but are still driven to stable values and included in the parity calculation.

Even parity is calculated for all PCI operations: the value of PCI*n*_PAR is generated such that the number of ones on PCI*n*_AD[31:0], PCI_C/$\overline{\text{BE}}$[3:0] and PCI*n*_PAR equals an even number. The PCI*n*_PAR signal is driven when the address/data lines are driven and follow the corresponding address or data by one clock.

The PCI controller checks the parity after all valid address phases (the assertion of $\overline{\text{PCI}n\text{_FRAME}}$) and for valid data transfers ($\overline{\text{PCI}n\text{_IRDY}}$ and $\overline{\text{PCI}n\text{_TRDY}}$ asserted) involving the PCI controller. When an address or data parity error is detected, the detected-parity-error bit in the configuration space status register is set (see Section 13.3.3.4, “PCI Status Configuration Register.”)

13.4.5.2 Error Reporting

Except for setting the detected-parity-error bit, all parity error reporting and response is controlled by the parity-error-response bit (see Section 13.3.3.3, “PCI Command Configuration Register,” for more information). If the parity-error-response bit is cleared, the PCI controller completes all transactions regardless of parity errors (address or data). If the bit is set, the PCI controller asserts $\overline{\text{PCI}n\text{_PERR}}$ two clocks after the actual data transfer in which a data parity error is detected, and keeps $\overline{\text{PCI}n\text{_PERR}}$ asserted for one clock. When acting as an initiator during a read transaction or as a target involved in a write to system memory the PCI controller asserts $\overline{\text{PCI}n\text{_PERR}}$.

Figure 13-52 shows the possible assertion points for PCI*n*_PERR if the PCI controller detects a data parity error.

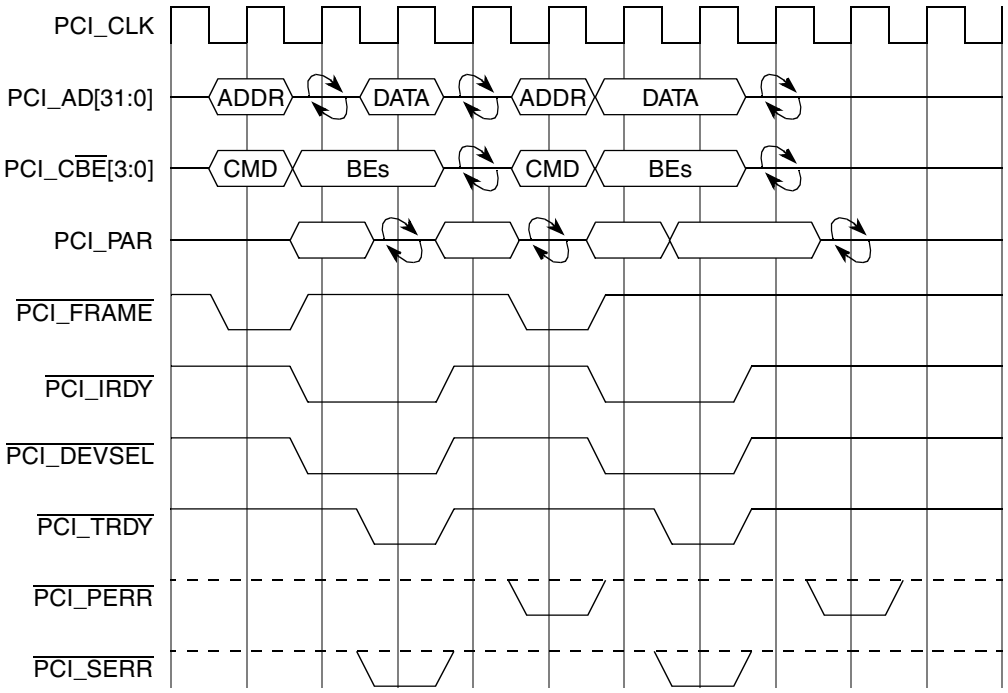


Figure 13-52. PCI Parity Operation

As an initiator, the PCI controller attempts to complete the transaction on the PCI bus if a data parity error is detected and sets the data-parity-reported bit in the configuration space status register. If a data parity error occurs on a read transaction, the PCI controller aborts the transaction internally. As a target, the PCI controller completes the transaction on the PCI bus even if a data parity error occurs. If parity error occurs during a write to system memory, the transaction completes on the PCI bus, but is aborted internally, insuring that potentially corrupt data does not go to memory.

When the PCI controller asserts PCI*n*_SERR, it sets the signaled-system-error bit in the configuration space status register. Additionally, if the error is an address parity error, the parity-error-detected bit is set; reporting an address parity error on PCI*n*_SERR is conditioned on the parity-error-response bit being enabled in the command register. PCI*n*_SERR is asserted when the PCI controller detects an address parity error while acting as a target. The system error is passed to the PCI controller's interrupt processing logic to assert MCP. Figure 13-52 shows where the PCI controller could detect an address parity error and assert PCI*n*_SERR or where the PCI controller, acting as an initiator, checks for the assertion of PCI*n*_SERR signaled by the target detecting an address parity error.

As a target that asserts PCI*n*_SERR on an address parity, the PCI controller completes the transaction on the PCI bus, aborting internally if the transaction is a write to system memory. If PCI*n*_PERR is asserted during a PCI controller write to PCI, the PCI controller attempts to continue the transfer, allowing the target to abort/disconnect if desired. If the PCI controller detects a parity error on a read from PCI, the PCI controller aborts the transaction internally and continues the transfer on the PCI bus, allowing the target to abort/disconnect if desired.

In all cases of parity errors on the PCI bus, regardless of the parity-error-response bit, information about the transaction is logged in the PCI error control capture register, the PCI error address capture register and the PCI error data capture register; $\overline{\text{MCP}}$ is also asserted to the core as an option.

13.4.6 PCI Inbound Address Translation

For inbound transactions (transactions generated by an external master on the PCI bus where the PCI controller responds as a slave device), the PCI controller only responds to PCI addresses within the windows mapped by the PCI inbound base address registers (PIBARs). If there is an address hit in one of the PIBARs, the PCI address is translated from PCI space to local memory space through the associated PCI inbound translation address registers (PITARs). This allows an external master to access local memory. Each PIBAR register is associated with a PITAR and PIWAR which are located in the PCI controller's PCI CSR space. Figure 13-53 shows an example translation window for inbound memory accesses.

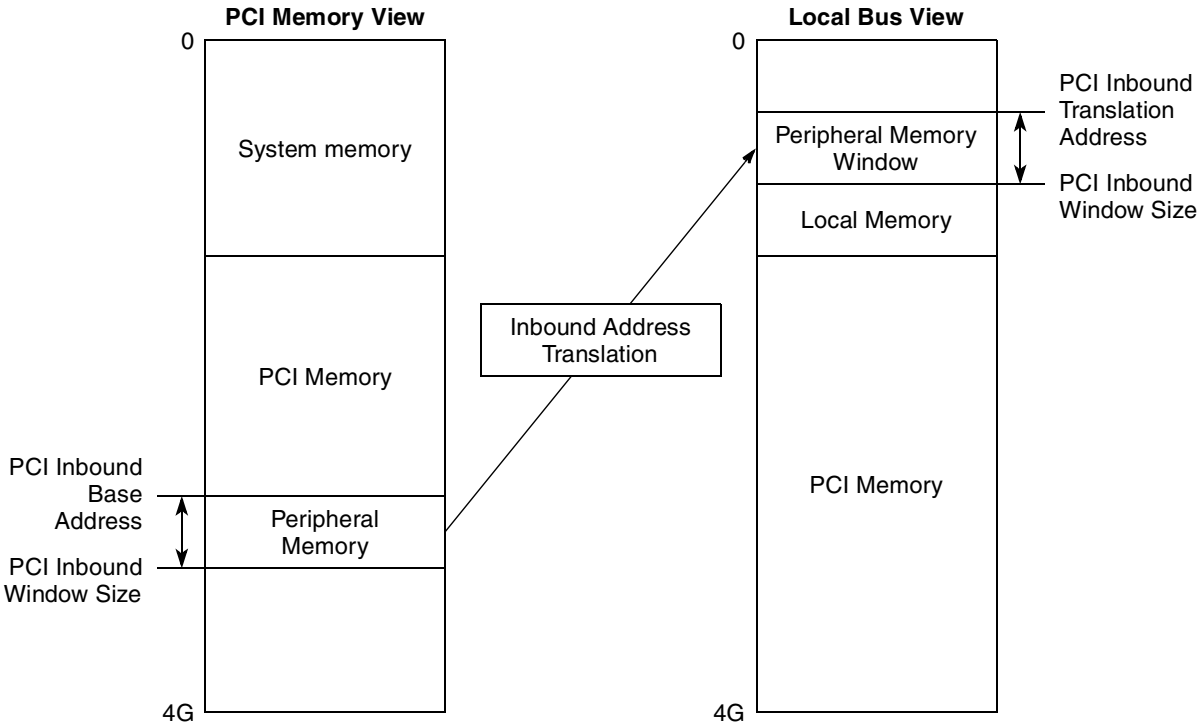


Figure 13-53. Inbound PCI Memory Address Translation

There are three full sets of inbound translation registers, in addition to the PIMMR base address register, allowing four simultaneous translation windows, one to a fixed destination and three programmable. Only two of the programmable windows can be mapped anywhere in the 64-bit PCI address space. Window 0 can only be mapped within the lowest 4-Gbyte space. Software can move the programmable translation base addresses during run-time to access different portions of local memory, but the PCI inbound translation windows may not overlap.

The translation windows are disabled after reset, that is, after reset, the PCI controller does not acknowledge externally mastered transactions on the PCI bus by asserting $\overline{\text{PCI}n\text{_DEVSEL}}$ until the inbound translation windows are enabled.

13.4.7 CompactPCI Hot Swap Specification Support

CompactPCI is an open specification supported by the PCI Industrial Computer Manufacturers Group (PICMG) and is intended for embedded applications using PCI. CompactPCI Hot Swap is an extension of the CompactPCI specification and allows the insertion and extraction (or “hot swapping”) of boards without adversely affecting system operation. The hot swap specification defines the following levels of support:

- Hot swap capable
- Hot swap friendly
- Hot swap ready

The PCI controller is hot swap friendly, meaning that it supports the hardware and software connection processes as defined in the hot swap specification. This level of support allows the board and system designers to build full Hot Swap and high availability systems based on the PCI controller as a PCI target device. For details on the hot swap process, refer to the *Hot Swap Specification PICMG 2.1*, R1.0, August 3, 1998.

13.5 Initialization/Application Information

The following sections describe initialization sequences for host and agent modes.

13.5.1 Initialization Sequence for Host Mode

The following sequences must be followed in host mode:

1. Enable PCI output clocks and select desired frequency ratios. See Section 4.4.1, “Clocking in PCI Host Mode.”
2. Wait for at least 1 ms to enable stable clocks into agent devices
3. Deactivate PCI_RESET_OUT signal for PCI1 and PCI2. See Table 13-3 for more information on PCI*n*_RESET_OUT signal.
4. Wait for at least 1 ms to enable devices to complete the powerup sequence.
5. Configure PCI internal registers and PCI agents to desired modes of operation

13.5.2 Initialization Sequence for Agent Mode

The following sequences must be followed in agent mode:

1. Optionally initialize subsystem vendor ID/device ID
   a) Initialize PCI inbound window size in PIWAR[1:3] desired window size
   b) Unlock configuration lock in PCI function configuration register

Chapter 14
Security Engine (SEC) 2.4

This chapter describes the functionality of the devices's integrated security engine (SEC 2.4). It addresses the following topics:

- Section 14.1, "Overview"
- Section 14.2, "Architecture Overview"
- Section 14.3, "Configuration of Internal Memory Space"
- Section 14.4, "Descriptor Overview"
- Section 14.5, "Execution Units"
- Section 14.6, "Crypto-Channels"
- Section 14.7, "Controller"
- Section 14.8, "Bus Interface"
- Section 14.9, "Power–Saving Mode"

14.1 Overview

The SEC 2.4 is designed to offload computationally intensive security functions, such as key generation and exchange, authentication, and bulk encryption from the device processor core. It is optimized to process all the algorithms associated with IPSec, IKE, SSL/TLS, iSCSI, SRTP, and 802.11i. The SEC 2.4 is derived from integrated security cores found in other members of the PowerQUICC family, including SEC 1.0, the version implemented in the MPC8272/MPC8248.

The security engine's execution units (EUs) and primary features include the following:

- PKEU—Public key execution unit that supports the following:
  - RSA and Diffie-Hellman algorithms
    - Programmable field size up to 2048 bits
  - Elliptic curve cryptography
    - F_2m and F(p) modes
    - Programmable field size up to 511 bits
- DEU—Data encryption standard execution unit
  - DES, 3DES algorithms
  - Two key (K1, K2) or three key (K1, K2, K3) for 3DES
  - ECB and CBC modes for both DES and 3DES
- AESU—Advanced encryption standard unit
  - Implements the Rijndael symmetric-key cipher

  - ECB, CBC, CCM, and counter modes
  - 128-, 192-, 256-bit key lengths
- AFEU—ARC four execution unit
  - Implements a stream cipher compatible with the RC4 algorithm
  - 40- to 128-bit programmable key
- MDEU—Message digest execution unit
  - SHA with 160-, 224-, or 256-bit message digest
  - MD5 with 128-bit message digest
  - HMAC with either algorithm
- RNG—Random number generator
- XOR parity generation accelerator for RAID applications
- Master/slave logic, with DMA capability
  - 32-bit address/64-bit data
  - Up to 200-MHz operation
  - Master interface allows multiple pipelined requests
  - DMA blocks can be on any byte boundary
- Four channels, each supporting a queue of commands (descriptor pointers)
  - Dynamic assignment of crypto-execution units through an integrated controller
  - 256-byte buffer FIFOs on data input and output paths of each execution unit, with flow control for large data sizes
- Scatter/Gather capability
  - Gather capability enables the SEC 2.4 to concatenate multiple segments of memory when reading input data
  - Similarly, scatter capability enables SEC 2.4 to write to multiple segments of memory when writing output data

14.2 Architecture Overview

The SEC 2.4 (henceforth referred to as SEC) can act as a master on the internal bus. This allows the SEC to alleviate the data movement bottleneck normally associated with slave-only cores. The host processor accesses the SEC through its device drivers, using system memory for data storage. The SEC resides in the peripheral memory map of the processor; therefore, when an application requires cryptographic functions, it simply creates descriptors for the SEC that define the cryptographic function to be performed and the location of the data. The SEC's bus-mastering capability permits the host processor to set up a crypto-channel with a few short register writes, leaving the SEC to perform reads and writes on system memory to complete the required task.

Figure 14-1 shows that the SEC communicates with other modules via the internal bus.

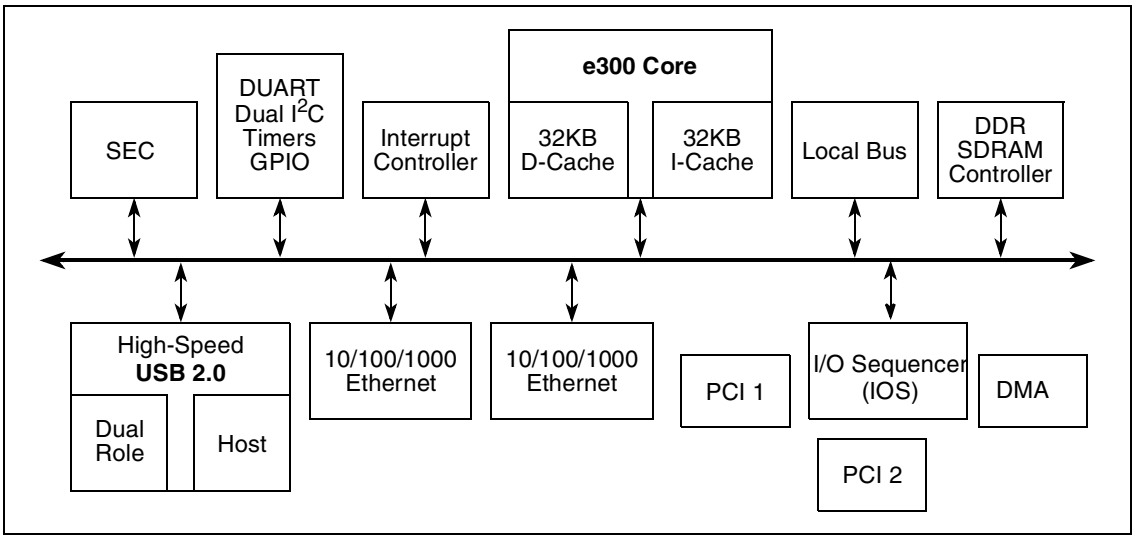


Figure 14-1. SEC Connected to MPC8349EA Internal Bus

A block diagram of the SEC internal architecture is shown in Figure 14-2. The bus interface module is designed to transfer 64-bit words between the bus and any register inside the SEC.

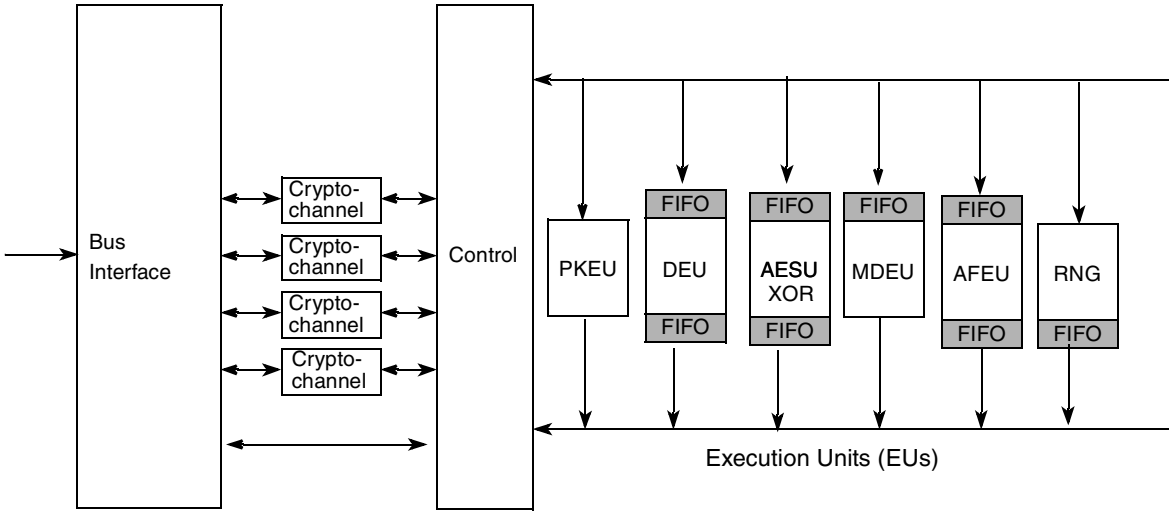


Figure 14-2. SEC Functional Modules

An operation begins when the host writes a descriptor pointer to the fetch FIFO in one of the four SEC channels. From this point on, the channel directs the sequence of operations. The channel uses the descriptor pointer to read the descriptor, then decodes the first word of the descriptor to determine the operation to be performed to select and the crypto-execution units needed to perform it. The channel requests the controller to assign the needed crypto-execution units. Next the channel requests that the controller fetch the keys, and data from locations specified in the rest of the descriptor. The controller satisfies the requests by making requests to the master interface following the programmable priority scheme. Data is fed into the execution units through their registers and input FIFOs. The execution units

read from their input FIFOs and write processed data to their output FIFOs. The channel requests the controller to write data from the output FIFOs and registers back to system memory through the master/slave interface.

For most packets, the entire payload is too long to fit in an execution unit's input or output FIFO. The SEC then uses a flow control scheme for reading and writing data. The channel directs the controller to read bursts of input as necessary to keep refilling the input FIFO, until the entire payload has been fetched. Similarly, the channel directs the controller to write bursts of output whenever enough accumulates in the execution unit's output FIFO.

14.2.1 Data Packet Descriptors

As a crypto-acceleration block, the SEC controller has been designed for easy use and integration with existing systems and software. All cryptographic functions are accessible through descriptors. A descriptor specifies a cryptographic function to be performed and contains pointers to all necessary input data and to the places where output data is to be written. Some descriptor types perform multiple functions to facilitate particular protocols. A data packet descriptor is diagrammed in Table 14-1.

Table 14-1. Example Data Packet Descriptor

Field Name	Value/Type	Description
DPD_DES_CTX_CRYPT	TBD	Representative header for DES using context to encrypt
LEN_CTXIN PTR_CTXIN	Length Pointer	Number of bytes to be written Pointer to context (IV) to be written into DES engine
LEN_KEY PTR_KEY	Length Pointer	Number of bytes in key Pointer to block cipher key
LEN_DATAIN PTR_DATAIN	Length Pointer	Number of bytes of data to be ciphered Pointer to data to perform cipher upon
LEN_DATAOUT PTR_DATAOUT	Length Pointer	Number of bytes of data after ciphering Pointer to location where cipher output is to be written
LEN_CTXOUT PTR_CTXOUT	Length Pointer	Length of output context (IV) Pointer to location where altered context is to be written
Null length Null pointer	Length Pointer	Zeros for fixed length descriptor filter Zeros for fixed length descriptor filter
Null length Null pointer	Length Pointer	Zeros for fixed length descriptor filter Zeros for fixed length descriptor filter

Each descriptor contains eight long-words (64 bits each), consisting of the following:

- One long-word of header—The header describes the required services and encodes information indicating which EUs to use and which modes to set. It also indicates if notification should be sent to the host when the descriptor operation is complete.
- Seven long-words containing pointers and lengths used to locate input or output data. Each pointer can either point directly to the data, or can point to a link table that lists a set of data segments to be concatenated.

For more information, refer to Section 14.4, "Descriptor Overview."

14.2.2 Execution Units (EUs)

Execution unit (EU) is the generic term for a functional block that performs the mathematical permutations required by protocols used in cryptographic processing. The EUs are compatible with IPSec, IKE, SSL/TLS, iSCSI, SRTP, and 802.11i processing, and can work together to perform high-level cryptographic tasks. The SEC's execution units are as follows:

- PKEU—For computing asymmetric-key operations, including modular exponentiation (and other modular arithmetic functions) or ECC point arithmetic
- DEU—For performing block cipher, symmetric-key cryptography using DES and 3DES
- AFEU—For performing RC-compatible stream cipher symmetric-key cryptography
- AESU—For performing the advanced encryption standard algorithm and XOR acceleration
- MDEU—For performing security hashing using MD-5, SHA-1, or SHA-256
- RNG—For random number generation

Each EU is described in detail in Section 14.5, "Execution Units."

14.2.2.1 Public Key Execution Unit (PKEU)

The PKEU can perform many advanced mathematical functions to support both RSA and ECC public-key cryptographic algorithms. ECC is supported in both F_2m (polynomial-basis) and F(p) modes. This EU supports all levels of functions to assist the host microprocessor to perform its desired cryptographic task. For example, at the highest level, the accelerator performs modular exponentiations to support RSA and performs point multiplies to support ECC. At the lower levels, the PKEU can perform simple operations such as modular multiplies. For more information, refer to Section 14.5.1, "Public Key Execution Unit (PKEU)."

14.2.2.1.1 Elliptic Curve Operations

The PKEU has its own data and control units, including a general-purpose register file in the programmable-size arithmetic unit. The field or modulus size can be programmed to any value between 160 and 512 bits in programmable increments of 8, with each value i supporting all actual field sizes from 8i – 7 to 8i. The result is hardware supporting a wide range of cryptographic security. Larger field/modulus sizes result in greater security but lower performance; processing time is determined by field or modulus size. For example, a field size of 160 is roughly equivalent to the security provided by 1024 bit RSA. A field size set to 208 roughly equates to 2048 bits of RSA security.

The PKEU contains routines implementing the atomic functions for elliptic curve processing—point arithmetic and finite field arithmetic. The point operations (multiplication, addition and doubling) involve one or more finite field operations, which are addition, multiplication, inverse, and squaring. Point add and double each use all four finite field operations. Similarly, point multiplication uses all EC point operations as well as the finite field operations. All these functions are supported both in modular arithmetic and polynomial basis finite fields.

14.2.2.1.2 Modular Exponentiation Operations

The PKEU can also perform ordinary integer modulo arithmetic. This arithmetic is an integral part of the RSA public key algorithm; however, it can also play a role in the generation of ECC digital signatures and Diffie-Hellman key exchanges.

Modular arithmetic functions supported by the SEC's PKEU include the following:

- R^2 mod N
- A'^E mod N
- (A × B) R^{-1} mod N
- (A × B) R^{-2} mod N
- (A + B) mod N
- (A – B) mod N

Where the following variable definitions: A′ = AR mod N, N is the modulus vector, A and B are input vectors, E is the exponent vector, R is 2^s, where s is the bit length of the N vector rounded up to the nearest multiple of 32.

The PKEU can perform modular arithmetic on operands up to 2048 bits in length. The modulus must be larger than or equal to 97 bits (13 bytes). This is not seen as a limitation since for sizes smaller than this, no useful cryptographic application exists. Furthermore, for small data sizes the overhead of using a hardware accelerator would not be justified. The PKEU uses the Montgomery modular multiplication algorithm to perform core functions. The addition and subtraction functions exist to help support known methods of the Chinese remainder theorem (CRT) for efficient implementation of the RSA algorithm.

14.2.2.2 Data Encryption Standard Execution Unit (DEU)

The DES execution unit (DEU) performs bulk data encryption/decryption, in compliance with the data encryption standard algorithm (ANSI x3.92). The DEU can also compute 3DES, an extension of the DES algorithm in which each 64-bit input block is processed three times. The SEC supports 2-key (K1=K3) or 3-key 3DES.

The DEU operates by permuting 64-bit data blocks with a shared 56-bit key and an initialization vector (IV). The SEC supports two modes of operation: electronic code book (ECB) and cipher clock chaining (CBC).

For more information, refer to Section 14.5.2, "Data Encryption Standard Execution Unit (DEU)."

14.2.2.3 ARC Four Execution Unit (AFEU)

The AFEU accelerates a bulk encryption algorithm compatible with the RC4 stream cipher from RSA Security, Inc. The algorithm is byte-oriented, meaning a byte of plain text is encrypted with a key to produce a byte of ciphertext. The key is variable length and the AFEU supports key lengths from 8 to 128 bits (in byte increments), providing a wide range of security strengths. ARC4 is a symmetric algorithm, meaning each of the two communicating parties share the same key.

For more information, refer to Section 14.5.3, "ARC Four Execution Unit (AFEU)."

14.2.2.4 Advanced Encryption Standard Execution Unit (AESU)

The AESU is used to accelerate bulk data encryption/decryption in compliance with the Advanced Encryption Standard (Rijndael) algorithm. The AESU executes on 128-bit blocks with a choice of key sizes: 128, 192, or 256 bits.

AESA is a symmetric-key algorithm; the sender and receiver use the same key for both encryption and decryption. The session key and IV are supplied to the AESU module prior to encryption. The processor supplies data to the module that is processed as a 128-bit input. The AESU operates in ECB, CBC, CTR, and CCM modes.

The AESU is also used for performing the eXclusive OR (XOR) operation used to generate parity data for RAID storage applications. When operating in this mode, no session keys are involved, and the AESU XORs up to 3 data streams at a time to produce parity data.

For more information, refer to Section 14.5.6, “Advanced Encryption Standard Execution Units (AESU).”

14.2.2.5 Message Digest Execution Unit (MDEU)

The MDEU computes a single message digest (or hash or integrity check) value of all the data presented on the input bus, using either the MD5, SHA-1 or SHA-256 algorithms for bulk data hashing. With any hash algorithm, the larger message is mapped onto a smaller output space; therefore, collisions are possible, albeit not probable. The 160-bit hash value is a sufficiently large space such that collisions are extremely rare. The security of the hash function is based on the difficulty of locating collisions. That is, it is computationally infeasible to construct two distinct but similar messages that produce the same hash output.

- The MD5 generates a 128-bit hash; the algorithm is specified in RFC 1321.
- SHA-1 is a 160-bit hash function, specified by the ANSI X9.30-2 and FIPS 180-1 standards.
- SHA-224 and SHA-256 are cryptographic hash functions that provide integrity protection against collision attacks.
- The MDEU also supports HMAC computations, as specified in RFC 2104.

For more information, refer to Section 14.5.4, “Message Digest Execution Unit (MDEU).”

14.2.2.6 Random Number Generator (RNG)

The RNG is a functional block capable of generating 64-bit random numbers. It is designed to comply with FIPS 140-1 standards for randomness and non-determinism.

Because many cryptographic algorithms use random numbers as a source for generating a secret value (a nonce), it is desirable to have a private RNG for use by the SEC. The anonymity of each random number must be maintained, as well as the unpredictability of the next random number. The FIPS-140 ‘common criteria’ compliant private RNG allows the system to develop random challenges or random secret keys. The secret key can thus remain hidden from even the high-level application code, providing an added measure of physical security.

For more information, refer to Section 14.5.5, “Random Number Generator (RNG).”

14.2.3 Crypto-Channels

The SEC includes four crypto-channels that manage data and EU function. Each channel consists of the following:

- A fetch FIFO, which holds a queue of pointers to descriptors waiting to be serviced
- A configuration register, which allows the user a number of options for SEC event signaling.
- Control registers containing information about the transaction in process
- A status register containing an indication of the last unfulfilled bus request
- A descriptor buffer memory used to store the active descriptor
- Scatter and gather link table buffer memory used to store the active link table

Whenever a channel is idle and its fetch FIFO is non-empty, the channel reads the next descriptor pointer from the fetch FIFO. Using this pointer, the channel fetches the descriptor and places it in its descriptor buffer. To service this descriptor, the channel directs execution of the following steps.

1. Analyze the descriptor header to determine the cryptographic services required and request use of the appropriate EU(s) from the controller.
2. Wait for the controller to grant access to the required EU(s).
3. Set the appropriate mode bits in the EU(s) for the required service.
4. Fetch ‘data parcels’ using pointers from the descriptor buffer, and place them in either an EU input FIFO or EU registers (as appropriate). The term data parcel refers here to any input or output of a cryptographic process, such as a key, hash result, input context, output context, or text-data. ‘Context’ refers to either an IV or other internal EU state that can be read out or loaded in. ‘Text-data’ refers to plaintext or ciphertext to be operated on.
5. If the data size is greater than EU FIFO size, continue fetching input data, and writing output data to memory.
6. Wait for EU(s) to complete processing.
7. Upon completion, unload results from output FIFOs and context registers and write them to external memory using pointers in the descriptor buffer.
8. If multiple services are requested, go back to step 3.
9. Release the EUs. (Note that in previous Freescale security co-processors, it was possible to reserve an EU for use on multiple descriptors. With the added capabilities in SEC 2.4, such ‘static’ assignments are no longer necessary and are not supported. EUs are always released at the completion of a descriptor.)
10. If ‘done notification’ is enabled in the descriptor header, perform this notification.

The channel can signal to the host that it is done with a descriptor via interrupt or by a writeback of the descriptor header into host memory. In the case of writeback, the value written back is identical to the header that was read, with the exception that a done field is set to all 1s. In addition, the channel can be configured to write back other status fields that indicate the result of ICV checking (if any). The user can opt to do this signaling at end of every descriptor, or at end of selected descriptors. For more about configuring signaling, see Section 14.6.1.1, “Crypto-Channel Configuration Register (CCCR).”

Many security protocols involve both encryption and hashing of packet payloads. To accomplish this without requiring two passes through the data, channels can configure data flows through more than one EU. In such cases, one EU is designated the primary EU, and the other as the secondary EU. The primary EU receives its data from memory through the controller, and the secondary EU receives its data by 'snooping' the SEC buses.

There are two types of snooping.

- Input data can be fed to the primary EU and the same input data snooped by the secondary EU. This is called 'in-snooping.'
- Output data from the primary EU can be snooped by the secondary EU. This is called 'out-snooping.'

In the SEC, the secondary EU is always the MDEU.

For more information, refer to Section 14.6, "Crypto-Channels."

14.2.4 SEC Controller

The SEC controller manages on-chip resources, including the individual execution units (EUs), FIFOs, the master/slave interface, and the internal buses that connect all the various modules. The controller receives service requests from the master/slave interface and various crypto-channels, and schedules the required activities. The controller can configure each of the on-chip resources in two modes:

- Host-controlled access—The host is directly responsible for all data movement into and out of the resource. This mode is typically only used in debug.
- Channel-controlled access—A channel can request a particular service from any available execution unit. This is the normal operating condition.

The system bus interface and access to system memory are critical factors in performance, and the 64-bit master/slave interface of the SEC controller allows it to achieve performance unattainable on secondary buses.

14.2.4.1 Host-Controlled Access

All execution units (EU) can be used entirely through register read/write access. The SEC operates as a slave, and the host must target write the information typically provided through the descriptor into the appropriate registers and FIFOs of the SEC. This mode is more CPU intensive, and requires a great deal of familiarity with the SEC registers. It is recommended that host-controlled access be used only for operations using a single EU, and for debug purposes.

For more information, refer to Section 14.7, "Controller."

14.2.4.2 Channel-Controlled Access

Processing begins when a descriptor pointer is written to the fetch FIFO of one of the channels. Based on the services requested by the descriptor header, the channel asks the controller to assign the necessary EUs to that channel. If all appropriate EUs are already reserved by other channels, the channel stalls and waits to fetch

data until an appropriate EU is available. If multiple channels simultaneously request the same EU, the EU is assigned on a weighted priority or round-robin basis.

Once the required EU has been reserved, the channel requests that the controller fetch and load the appropriate data. The controller acts as a master on the system bus, reading and writing on byte boundaries. The channel operates the EU, and makes further requests to the controller to write output data to system memory. When the descriptor processing is complete, the channel asks the controller to release the EU for use by other channels.

14.2.5 Bus Interface

The master/slave interface manages communication between the SEC's internal execution units and the device internal bus. All on-chip resources are memory mapped, and the slave accesses to the SEC may be addressed on byte boundaries. The SEC performs initiator reads on byte boundaries and internally adjusts the data to place on word boundaries as appropriate. Access to system memory is a critical factor in performance, and the 64-bit master/slave interface of the SEC allows it to achieve performance unattainable on secondary buses.

For more information, refer to Section 14.8, "Bus Interface."

14.3 Configuration of Internal Memory Space

Table 14-2 shows the base address map, while Table 14-3 provides the address map, including all registers in the execution units. The 18-bit SEC address bus value is shown. These address values are offsets from the internal memory mapped registers base address, as programmed in the IMMRBAR. See Section 2.3, "Complete IMMR Map," for more information.

Note that these tables show modulo-8 addresses; the three least significant address bits that are used to select bytes within 64-bit-words are not shown.

Table 14-2. SEC Base Address Map

Offset	Module	Description	Type	Section/Page
0x3_0000–0x3_0FFF	—	Reserved	—	—
0x3_1000–0x3_10FF	Controller	Arbiter/controller control register space	Resource control	14.7/14-93
0x3_1100–0x3_11FF	Channel_1	Crypto-channel 1	Data control	14.6/14-81
0x3_1200–0x3_12FF	Channel_2	Crypto-channel 2		
0x3_1300–0x3_13FF	Channel_3	Crypto-channel 3		
0x3_1400–0x3_14FF	Channel_4	Crypto-channel 4		

Table 14-2. SEC Base Address Map (continued)

Offset	Module	Description	Type	Section/Page
0x3_2000–0x3_2FFF	DEU	DES/3DES execution unit	Crypto EU	14.5.2/14-34
0x3_4000–0x3_4FFF	AESU	AES execution unit		14.5.6/14-68
0x3_6000–0x3_6FFF	MDEU	Message digest execution unit		14.5.4/14-51
0x3_8000–0x3_8FFF	AFEU	ARC four execution unit		14.5.3/14-42
0x3_A000–0x3_AFFF	RNG	Random number generator		14.5.5/14-63
0x3_C000–0x3_CFFF	PKEU	Public key execution unit		14.5.1/14-26

Table 14-3 shows the system address map showing all functional registers. Undefined 4-byte address spaces within offset 0x000–0xFFF are reserved.

Table 14-3. SEC Address Map

Offset	Register	Access	Reset	Section/Page
Controller Registers				
0x3_1008	IMR—Interrupt mask register	R/W	0x0000_0000_0000_0000	14.7.2.1/14-94
0x3_1010	ISR—Interrupt status register	R	0x0000_0000_0000_0000	14.7.2.2/14-96
0x3_1018	ICR—Interrupt clear register	W	0x0000_0000_0000_0000	14.7.2.3/14-96
0x3_1020	ID—Identification register	R	0x0000_0000_0000_0040	14.7.2.4/14-98
0x3_1028	EUASR—EU assignment status register	R	0xF0F0_F0F0_00FF_F0F0	14.7.2/14-93
0x3_1030	MCR—Master control register	R/W	0000_0000_0000_0000	14.7.2.5/14-98
Channel 1				
0x3_1108	CCCR1—Crypto-channel 1 configuration register	R/W	0x0000_0000_0000_0000	14.6.1.1/14-82
0x3_1110	CCPSR1—Crypto-channel 1 pointer status register	R	0x0000_0000_0000_0007	14.6.1.2/14-85
0x3_1140	CDPR1—Crypto-channel 1 current descriptor pointer register	R	0x0000_0000_0000_0000	14.6.1.3/14-90
0x3_1148	FF1—Crypto-channel 1 fetch FIFO address register	W	0x0000_0000_0000_0000	14.6.1.4/14-90
0x3_1180–0x3_11BF	DBn—Crypto-channel 1descriptor buffers[0–7]	R	0x0000_0000_0000_0000	14.6.1.5/14-91
Channel 2				
0x3_1208	CCCR2—Crypto-channel 2 configuration register	R/W	0x0000_0000_0000_0000	14.6.1.1/14-82
0x3_1210	CCPSR2—Crypto-channel 2 pointer status register	R	0x0000_0000_0000_0007	14.6.1.2/14-85
0x3_1240	CDPR2—Crypto-channel 2 current descriptor pointer register	R	0x0000_0000_0000_0000	14.6.1.3/14-90
0x3_1248	FF2—Crypto-channel 2 fetch FIFO address register	W	0x0000_0000_0000_0000	14.6.1.4/14-90
0x3_1280–0x3_12BF	DBn—Crypto-channel 2 descriptor buffers[0–7]	R	0x0000_0000_0000_0000	14.6.1.5/14-91

Table 14-3. SEC Address Map (continued)

Offset	Register	Access	Reset	Section/Page
Channel 3				
0x3_1308	CCCR3—Crypto-channel 3 configuration register	R/W	0x0000_0000_0000_0000	14.6.1.1/14-82
0x3_1310	CCPSR3—Crypto-channel 3 pointer status register	R	0x0000_0000_0000_0007	14.6.1.2/14-85
0x3_1340	CDPR3—Crypto-channel 3 current descriptor pointer register	R	0x0000_0000_0000_0000	14.6.1.3/14-90
0x3_1348	FF3—Crypto-channel 3 fetch FIFO address register	W	0x0000_0000_0000_0000	14.6.1.4/14-90
0x3_1380–0x3_13BF	DBn—Crypto-channel 3 descriptor buffers[0–7]	R	0x0000_0000_0000_0000	14.6.1.5/14-91
Channel 4				
0x3_1408	CCCR4—Crypto-channel 4 configuration register	R/W	0x0000_0000_0000_0000	14.6.1.1/14-82
0x3_1410	CCPSR4—Crypto-channel 4 pointer status register	R	0x0000_0000_0000_0007	14.6.1.2/14-85
0x3_1440	CDPR4—Crypto-channel 4 current descriptor pointer register	R	0x0000_0000_0000_0000	14.6.1.3/14-90
0x3_1448	FF4—Crypto-channel 4 fetch FIFO address register	W	0x0000_0000_0000_0000	14.6.1.4/14-90
0x3_1480–0x3_14BF	DBn—Crypto-channel 4 descriptor buffers[0–7]	R	0x0000_0000_0000_0000	14.6.1.5/14-91
Data Encryption Standard Execution Unit (DEU)				
0x3_2000	DEUMR—DEU mode register	R/W	0x0000_0000_0000_0000	14.5.2.1/14-35
0x3_2008	DEUKSR—DEU key size register	R/W	0x0000_0000_0000_0000	14.5.2.2/14-36
0x3_2010	DEUDSR—DEU data size register	R/W	0x0000_0000_0000_0000	14.5.2.3/14-36
0x3_2018	DEURCR—DEU reset control register	R/W	0x0000_0000_0000_0000	14.5.2.4/14-37
0x3_2028	DEUSR—DEU status register	R	0x0000_0000_0000_0000	14.5.2.5/14-37
0x3_2030	DEUISR—DEU interrupt status register	R	0x0000_0000_0000_0000	14.5.2.6/14-38
0x3_2038	DEUICR—DEU interrupt control register	R/W	0x0000_0000_0000_3000	14.5.2.7/14-40
0x3_2050	DEUEUG—DEU EU-Go register	W	0x0000_0000_0000_0000	14.5.2.8/14-41
0x3_2100	DEUIV—DEU initialization vector register	R/W	0x0000_0000_0000_0000	14.5.2.9/14-42
0x3_2400	DEUK1—DEU key register 1	W	—	14.5.2.10/14-42
0x3_2408	DEUK2—DEU key register 2	W	—	14.5.2.10/14-42
0x3_2410	DEUK3—DEU key register 3	W	—	14.5.2.10/14-42
0x3_2800–0x3_2FFF	DEU FIFO	R/W	0x0000_0000_0000_0000	14.5.2.11/14-42
Advanced Encryption Standard Execution Unit (AESU)				
0x3_4000	AESUMR—AESU mode register	R/W	0x0000_0000_0000_0000	14.5.6.1/1414-68
0x3_4008	AESUKSR—AESU key size register	R/W	0x0000_0000_0000_0000	14.5.6.2/1414-71

Table 14-3. SEC Address Map (continued)

Offset	Register	Access	Reset	Section/Page
0x3_4010	AESUDSR—AESU data size register	R/W	0x0000_0000_0000_0000	14.5.6.3/1414-71
0x3_4018	AESURCR—AESU reset control register	R/W	0x0000_0000_0000_0000	14.5.6.4/14-72
0x3_4028	AESUSR—AESU status register	R	0x0000_0000_0000_0000	14.5.6.5/14-73
0x3_4030	AESUISR—AESU interrupt status register	R	0x0000_0000_0000_0000	14.5.6.6/14-74
0x3_4038	AESUICR—AESU interrupt control register	R/W	0x0000_0000_0000_1000	14.5.6.7/14-75
0x3_4050	AESUEMR—AESU end of message register	W	0x0000_0000_0000_0000	14.5.6.8/14-76
0x3_4100	AESU context memory registers	R/W	0x0000_0000_0000_0000	14.5.6.9/14-77
0x3_4400–0x3_4408	AESU key memory	R/W	0x0000_0000_0000_0000	14.5.6.9.5/14-81
0x3_4800–0x3_4FFF	AESU FIFO	R/W	0x0000_0000_0000_0000	14.5.6.9.6/14-81
Message Digest Execution Unit (MDEU)				
0x3_6000	MDEUMR—MDEU mode register	R/W	0x0000_0000_0000_0000	14.5.4.1/14-51
0x3_6008	MDEUKSR—MDEU key size register	R/W	0x0000_0000_0000_0000	14.5.4.3/14-55
0x3_6010	MDEUDSR—MDEU data size register	R/W	0x0000_0000_0000_0000	14.5.4.4/14-56
0x3_6018	MDEURCR—MDEU reset control register	R/W	0x0000_0000_0000_0000	14.5.4.5/14-56
0x3_6028	MDEUSR—MDEU status register	R	0x0000_0000_0000_0000	14.5.4.6/14-57
0x3_6030	MDEUISR—MDEU interrupt status register	R	0x0000_0000_0000_0000	14.5.4.7/14-58
0x3_6038	MDEUICR—MDEU interrupt control register	R/W	0x0000_0000_0000_1000	14.5.4.8/14-59
0x3_6040	MDEUICVSR—MDEU ICV size register	R/W	0x0000_0000_0000_0000	14.5.4.9/14-60
0x3_6050	MDEUEUG—MDEU EU-Go register	W	0x0000_0000_0000_0000	14.5.4.10/14-61
0x3_6100–0x3_6120	MDEU context memory registers	R/W	0x0000_0000_0000_0000	14.5.4.11/14-61
0x3_6400–0x3_647F	MDEU key memory	W	0x0000_0000_0000_0000	14.5.4.12/14-62
0x3_6800–0x3_6FFF	MDEU FIFO	W	0x0000_0000_0000_0000	14.5.4.13/14-63
ARC Four Execution Unit (AFEU)				
0x3_8000	AFEUMR—AFEU mode register	R/W	0x0000_0000_0000_0000	14.5.3.1/14-43
0x3_808	AFEUKSR—AFEU key size register	R/W	0x0000_0000_0000_0000	14.5.3.3/14-44
0x3_8010	AFEUDSR—AFEU data size register	R/W	0x0000_0000_0000_0000	14.5.3.4/14-45
0x3_8018	AFEURCR—AFEU reset control register	R/W	0x0000_0000_0000_0000	14.5.3.5/14-46
0x3_8028	AFEUSR—AFEU status register	R	0x0000_0000_0000_0000	14.5.3.6/14-46
0x3_8030	AFEUISR—AFEU interrupt status register	R	0x0000_0000_0000_0000	14.5.3.7/14-47

Table 14-3. SEC Address Map (continued)

Offset	Register	Access	Reset	Section/Page
0x3_8038	AFEUICR—AFEU interrupt control register	R/W	0x0000_0000_0000_1000	14.5.3.8/14-49
0x3_8050	AFEUEMR—AFEU end of message register	W	0x0000_0000_0000_0000	14.5.3.9/14-50
0x3_8100–0x3_81FF	AFEU context memory registers	R/W	0x0000_0000_0000_0000	14.5.3.10.1/14-50
0x3_8200	AFEU context memory pointers	R/W	0x0000_0000_0000_0000	14.5.3.10.2/14-51
0x3_8400	AFEUK0—AFEU key register 0	W	—	14.5.3.11/14-51
0x3_848	AFEUK1—AFEU key register 1	W	—	14.5.3.11/14-51
0x3_8800–0x3_8FFF	AFEU FIFO	R/W	0x0000_0000_0000_0000	14.5.3.11.1/14-51
Random Number Generator (RNG)				
0x3_A000	RNGMR—RNG mode register	R/W	0x0000_0000_0000_0000	14.5.5.1/14-63
0x3_A010	RNGDSR—RNG data size register	R/W	0x0000_0000_0000_0000	14.5.5.2/14-64
0x3_A018	RNGRCR—RNG reset control register	R/W	0x0000_0000_0000_0000	14.5.5.3/14-65
0x3_A028	RNGSR—RNG status register	R	0x0000_0000_0000_0000	14.5.5.4/14-65
0x3_A030	RNGISR—RNG interrupt status register	R	0x0000_0000_0000_0000	14.5.5.5/14-66
0x3_A038	RNGICR—RNG interrupt control register	R/W	0x0000_0000_0000_1000	14.5.5.6/14-67
0x3_A050	RNGEUG—RNG EU-Go register	W	0x0000_0000_0000_0000	14.5.5.7/14-68
0x3_A800–0x3_AFFF	RNG FIFO	R	0x0000_0000_0000_0000	14.5.5.8/14-68
Public Key Execution Unit (PKEU)				
0x3_C000	PKEUMR—PKEU mode register	R/W	0x0000_0000_0000_0000	14.5.1.1/14-26
0x3_C008	PKEUKSR—PKEU key size register	R/W	0x0000_0000_0000_0000	14.5.1.2/14-28
0x3_C010	PKEUDSR—PKEU data size register	R/W	0x0000_0000_0000_0000	14.5.1.3/14-28
0x3_C018	PKEURCR—PKEU reset control register	R/W	0x0000_0000_0000_0000	14.5.1.5/14-29
0x3_C028	PKEUSR—PKEU status register	R	0x0000_0000_0000_0000	14.5.1.6/14-30
0x3_C030	PKEUISR—PKEU interrupt status register	R	0x0000_0000_0000_0000	14.5.1.7/14-31
0x3_C038	PKEUICR—PKEU interrupt control register	R/W	0x0000_0000_0000_1000	14.5.1.8/14-32
0x3_C040	PKEUABS—PKEU AB size register	R/W	0x0000_0000_0000_0000	14.5.1.3/14-28
0x3_C050	PKEUEUG—PKEU EU-Go	W	0x0000_0000_0000_0000	14.5.1.9/14-33

Table 14-3. SEC Address Map (continued)

Offset	Register	Access	Reset	Section/Page
0x3_C200–0x3_C23F	PKEU parameter memory A0	R/W	0x0000_0000_0000_0000	14.5.1.10/14-34
0x3_C240–0x3_C27F	PKEU parameter memory A1	R/W	0x0000_0000_0000_0000	
0x3_C280–0x3_C2BF	PKEU parameter memory A2	R/W	0x0000_0000_0000_0000	
0x3_C2C0–0x3_C2FF	PKEU parameter memory A3	R/W	0x0000_0000_0000_0000	
0x3_C300–0x3_C33F	PKEU parameter memory B0	R/W	0x0000_0000_0000_0000	
0x3_C340–0x3_C37F	PKEU parameter memory B1	R/W	0x0000_0000_0000_0000	
0x3_C380–0x3_C3BF	PKEU parameter memory B2	R/W	0x0000_0000_0000_0000	
0x3_C3C0–0x3_C3FF	PKEU parameter memory B3	R/W	0x0000_0000_0000_0000	
0x3_C400–0x3_C4FF	PKEU parameter memory E	W	0x0000_0000_0000_0000	
0x3_C800–0x3_C8FF	PKEU parameter memory N	R/W	0x0000_0000_0000_0000	

14.4 Descriptor Overview

The host processor maintains a record of current secure sessions and the corresponding keys and contexts of those sessions. Once the host has determined that a security operation is required, it creates a descriptor containing all the information the SEC needs to perform the security operation. The host creates the descriptor in main memory, then writes a pointer to the descriptor into the fetch FIFO of one of the SEC channels. The channel uses this pointer to read the descriptor into its descriptor buffer. Once it obtains the descriptor, the SEC uses its bus mastering capability to obtain inputs and write results, thus off-loading data movement and encryption operations from the host processor.

For test purposes, it is also possible for the host to write keys, context, and text-data directly to the SEC, using SEC's host-controlled mode. This method avoids use of descriptors.

14.4.1 Descriptor Structure

SEC descriptors are conceptually similar to descriptors used by most devices with DMA capability. The descriptors have a fixed length of 64 bytes, i.e. eight 64-bit words (referred to as dwords). A descriptor consists of one header dword and seven pointer dwords, as shown in Figure 14-3.

The header dword specifies the security operation to be performed, the execution unit(s) needed, and the modes for each execution unit. The pointer dwords, all of which have the same format, contain pointer and

	0–15	16	17–23	24–31	32–63
Header Dword	Header				Reserved
Pointer Dword 0	Length0	J0	Extent0	—	Pointer0
Pointer Dword 1	Length1	J1	Extent1	—	Pointer1
Pointer Dword 2	Length2	J2	Extent2	—	Pointer2
Pointer Dword 3	Length3	J3	Extent3	—	Pointer3
Pointer Dword 4	Length4	J4	Extent4	—	Pointer4
Pointer Dword 5	Length5	J5	Extent5	—	Pointer5
Pointer Dword 6	Length6	J6	Extent6	—	Pointer6

Figure 14-3. Descriptor Format

length information for locating input or output data parcels (such as keys, context, or text-data). The large number of pointers provided in the descriptor allows for multi-algorithm operations that require fetching of multiple keys, as well as fetch and return of contexts. Any pointer double word that is not needed can be given a length of zero and the channel skips over the corresponding operations.

SEC descriptors include scatter/gather capability, which means that each pointer in a descriptor can be either a direct pointer to a contiguous parcel of data, or can be a pointer to a ‘link table’ which is a list of pointers and lengths used to assemble the data parcel. When a link table is used to read input data, this is referred to as a gather operation; when used to write output data, it is referred to as a scatter operation.

14.4.2 Descriptor Format—Header Dword

Descriptors are created by the host to guide the SEC through required cryptographic operations. The descriptor header defines the operations to be performed, the mode for each operation, and internal addressing used by the controller and channel for internal data movement. The SEC device drivers allow the host to create proper headers for each cryptographic operation.

	0–3	4–11	12–15	16–23	24–28	29	30	31
Field	OP_0		OP_1		DESC_TYPE	—	DIR	DN
	EU_SEL0	MODE0	EU_SEL1	MODE1				

	32–63
Field	—

Figure 14-4. Header Dword

Table 14-4. Header Dword Bit Definitions

Bits	Name	Description
OP_0		
0–3	EU_SEL0	Primary EU select: See 14.4.2.1/14-17 for possible values.
4–11	MODE0	Primary mode: Mode data used to program the primary EU. The mode data is to the chosen EU. This field is passed directly to bits 56–63 of the mode register in the selected EU.
OP_1		
12–15	EU_SEL1	Secondary EU select: See 14.4.2.1/14-17 for possible values.
16–23	MODE1	Secondary mode: Mode data used to program the primary EU. The mode data is to the chosen EU. This field is passed directly to bits 56–63 of the mode register in the selected EU.
24–28	DESC_TYPE	Descriptor type. This field, along with the DIR field, determines the sequence of actions to be performed by the channel and selected EUs using the blocks of data listed in the rest of the descriptor. The attributes determined include the direction of data flow for each data block, which EU (primary or secondary) is accessed, what snooping options are used, and which internal EU addresses are accessed. See 14.4.2.2/14-18 for possible values.
29	—	Reserved.
30	DIR	Direction of overall data flow 0 Outbound 1 Inbound This field, along with the DESC_TYPE field, helps determine the sequence of actions to be performed by the channel and selected EUs.
31	DN	Done notification 0 No done notification. 1 Signal done to the host on completion of this descriptor. The done notification can take the form of an interrupt, or a modified header writeback, or both, depending upon the states of the channel done interrupt enable (CDIE) and channel done writeback enable (CDWE) bits in the channel configuration register. When writeback is used, the value written back is 0xFF in bits 0–7 of this long word.

14.4.2.1 Selecting Execution Units—EU_SEL0 and EU_SEL1

Table 14-5 shows the values for EU_SEL0 and EU_SEL1 in the descriptor header. The following rules govern the choices for these fields:

1. EU_SEL0 values of 0000 (no EU selected) or 0111–1111 (reserved) results in an unrecognized header error condition during processing of the descriptor header.
2. The only valid choices for EU_SEL1 are 0000 (No EU selected) or 0011 (MDEU). Any other choice causes an unrecognized header error condition.
3. If EU_SEL1 is 0011 (MDEU), then EU_SEL0 must be 0010 (DEU), 0110 (AESU), or 0001 (AFEU). All other values of EU_SEL0 causes an unrecognized header error condition.

Table 14-5. EU_SEL1 and EU_SEL2 Values

Value (binary)	Selected EU
0000	No EU selected
0001	AFEU
0010	DEU
0011	MDEU
0100	RNG
0101	PKEU
0110	AESU
0111–1110	Reserved
1111	Reserved for header writeback

14.4.2.2 Selecting Descriptor Type—DESC_TYPE

Table 14-6 shows the permissible values for the DESC_TYPE field in the descriptor header. Descriptor types from the SEC 1.0, which have zero in the last bit (xxxx_0), are listed first, followed by new SEC 2.4 types, which have one in the last bit (xxxx_1).

Table 14-6. Descriptor Types

Value (binary)	Descriptor Type	Notes
0000_0	aesu_ctr_nonsnoop	AESU CTR nonsnooping [1]
0001_0	common_nonsnoop_no_afeu	Common, nonsnooping, non-PKEU, non-AFEU [1]
0010_0	hmac_snoop_no_afeu	Snooping, HMAC, non-AFEU [2]
0011_0	—	Reserved
0100_0	—	Reserved
0101_0	common_nonsnoop_afeu	Common, nonsnooping, AFEU
0110_ 0	—	Reserved
0000_0	aesu_ctr_nonsnoop	AESU CTR nonsnooping
0001_0	common_nonsnoop	Common, nonsnooping, non-PKEU, non-AFEU
0010_0	hmac_snoop_no_afeu	Snooping, HMAC, non-AFEU
0011_0	—	Reserved
0100_0	—	Reserved
0101_0	common_nonsnoop_afeu	Common, nonsnooping, AFEU
0110_ 0	—	Reserved
0111_0	—	Reserved
1000_0	pkeu_mm	PKEU-Montgomery Multiplication

Table 14-6. Descriptor Types (continued)

Value (binary)	Descriptor Type	Notes
1001_0	—	Reserved
1010_0	—	Reserved
1011_ 0	—	Reserved
1100_0	hmac_snoop_aesu_ctr	AESU CTR hmac snooping [2]
1101_0	—	Reserved
1110_0	—	Reserved
1111_0	—	Reserved
0000_1	ipsec_esp	IPsec ESP mode encryption and hashing
0001_1	802.11i AES ccmp	CCMP encryption and hashing, suitable for 802.11i
0010_1	srtp	SRTP encryption and hashing
0011_1	pkeu_assemble	pkeu_assemble Elliptic Curve Cryptography
0100_1	pkeu_ptmul	pkeu_ptmul Elliptic Curve Cryptography
0101_1	pkeu_ptadd_dbl	pkeu_ptadd_dbl Elliptic Curve Cryptography
others	—	Reserved
0110_1	—	Reserved
0111_1	—	Reserved
1000_1	tls_ssl_block	TLS/SSL generic block cipher
1001_1	tls_ssl_stream	TLS/SSL generic stream cipher
1010_1	raid_xor	XOR data streams
All others	—	Reserved

[1] Type 0000_0 is for AES-CTR operations. Type 0001_0 also supports AES-CTR; however, to use AES-CTR with 0001_0, the user must prepend zeros to the AES ctx before loading the AES context registers.

[2] Type 1100_0 is for AES-CTR operations with HMAC. Type 0010_0 also supports AES-CTR with HMAC; however, to use AES-CTR with 0010_0, the user must prepend zeros to the AES ctx before loading the AES context registers.

For more about descriptor types and the data used for each type, see Section 14.4.5, “Descriptor Types.”

14.4.3 Descriptor Format—Pointer Dwords

The descriptor contains seven pointer dwords that define where in memory the SEC should access its input and output data parcels. The channel determines how it will use each of the pointer dwords based on the descriptor type and direction fields in the header. The channel accesses the first data parcel by starting at a location given by a POINTER value, and accessing a number of bytes given by a LENGTH or EXTENT value. Subsequent data parcels may be accessed by staring where a previous data parcel ended, or by starting at a different POINTER. The LENGTH or EXTENT used with any POINTER may be from the same pointer dword or from a different pointer dword in the same descriptor. Although the EXTENT field

exists in each pointer dword of the SEC descriptor, (only the EXTENTs in pointer dwords 3, 4, and 5 are currently in use).

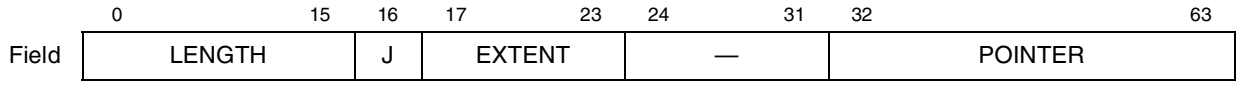


Figure 14-5. Pointer Dword

Table 14-7. Pointer Dword Field Definitions

Bits	Name	Description
0–15	LENGTH	Length. A number of bytes in the range 0 to 65535. The use of this field depends on the descriptor type and direction in the header dword. A value of zero causes the channel to skip this dword.
16	J	Jump. Determines whether to jump to a link table whenever the POINTER field in this same word is used. 0 The POINTER field points to data. 1 The POINTER field points to a link table, and scatter/gather is enabled.
17–23	EXTENT	Extent. A number of bytes in the range 0 to 127. The use of this field depends on the descriptor type and direction in the header dword.
24–31	—	Reserved
32–63	POINTER	Pointer. A memory address.

On occasion, a descriptor field may not be applicable to the requested service. With seven length/pointer pairs, it is possible that not all descriptor fields are required to load the required keys, context, and text-data. (Some operations, for example, do not require context.) Therefore, when processing descriptors the SEC skips entirely any pointer that has an associated LENGTH of zero.

Some descriptors involve more than seven parcels of input and output data. In these cases, it is necessary to use one POINTER field to address a sequence of data parcels.

LENGTH and EXTENT fields normally specify the sizes of data parcels. In some cases, however, the POINTER field is zero, and the LENGTH and/or EXTENT fields simply specify values to be written to an EU.

If the channel procedure calls for reading a parcel using a nonzero LENGTH field, but the POINTER field is zero, the length value is written to the EU but no data parcel is fetched from the bus.

The J bit in each pointer dword is used to enable the scatter/gather feature. If a data parcel to be read or written by SEC is in one contiguous block of memory locations, then the scatter/gather feature is not needed. In this case the POINTER should be set to point directly at the first byte of the parcel, and the J bit should be 0. On the other hand, if the data parcel is stored in several separate segments of memory, then the scatter/gather capability is needed to assemble or distribute the complete parcel. In this case the POINTER should be set to point to a link table, and the J bit should be 1. For link table format, see Section 14.4.4, “Link Table Format.”

Scatter/gather capability is available for all pointer dwords of all descriptor types, with the following exception(s):

- Raid-XOR descriptor type does not allow scatter/gather.

14.4.4 Link Table Format

Link tables implement scatter/gather capability. For gather operations, a link table specifies a list of memory segments that are to be concatenated in the process of assembling data parcels. For scatter operations, a link table specifies a list of memory segments into which the output data should be written. Scatter or gather of a data parcel may be specified by a single link table or by a chain of link tables that are linked together with pointers (see Figure 14-7).

The link table or chain of link tables accessed through some descriptor POINTER must specify enough memory segments to hold all the data that will be accessed through that pointer. In most cases, only a single data parcel is accessed through a given POINTER, and the chain of link tables specifies just that parcel. In other cases, the descriptor POINTER is used multiple times to access a sequence of data parcels, and the chain of link tables must supply data for the entire sequence. If a link table is used to access a sequence of data parcels, the end of each parcel must also be at the end of a memory segment. In other words, a single memory segment must not straddle two data parcels. An example of proper construction of link tables is illustrated in Figure 14-7.

A link table may contain any number of long word entries. There are two kinds of entries, ‘regular’ entries and ‘next’ entries. Each ‘regular’ entry specifies a memory segment by means of a 32-bit starting address (SEGPTR) and a 16-bit length (SEGLEN). A ‘next’ entry is used at the end of a link table to specify that the list of memory segments is continued in another link table. In a ‘next’ entry, the N bit is set and the SEGPTR field gives the address of the next link table, and the SEGLEN field must be 0. A chain of link tables may contain any number of link tables.

Whether the list of memory segments is in a single link table or split into several link tables, the last entry in the last link table is a ‘regular’ entry with the R (return) bit set. The R bit signifies the end of link table operations so that the channel returns to the descriptor for its next pointer (if any). Link tables are illustrated in Figure 14-7.

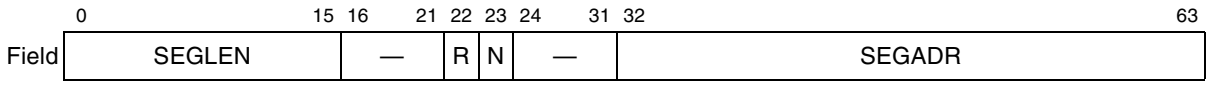


Figure 14-6. Link Table Entry Format

Table 14-8. Link Table Field Definitions

Bits	Name	Description
0–15	SEGLEN	Length. When N=0, a number in the range 1 to 65535, specifying the number of bytes in the memory segment. pointed to by SEGADR. A value of 0 will cause an error bit to be set in the Channel Pointer Status Register—GER for a gather operation or SER for a scatter operation (see 14.6.1.1/14-82). When N=1, must be zero.
16–21	—	Reserved
22	R	Return: When N=0: 0 No special action. 1 This is the last entry in the chain of link tables. If this entry does not specify the right number of bytes to complete the last data parcel, a GER or SER error will be set in the Channel Pointer Status Register (see 14.6.1.1/14-82). When N=1: ignored.

Table 14-8. Link Table Field Definitions (continued)

Bits	Name	Description
23	N	Next 0 No special action. 1 This is the last long word in the current link table. The SEGADR field is the address of the next link table in the chain.
24–31	—	Reserved
32–63	SEGADR	Segment address. A memory address.

For any sequence of data parcels accessed by a link table or chain of link tables, the combined lengths of the parcels (the sum of their LENGTH and/or EXTENT fields) must equal the combined lengths of the link table memory segments (SEGLEN fields). Otherwise the channel sets the appropriate error bit in the channel pointer status register—GER for a gather error or SER for a scatter error. See Section 14.6.1.1, “Crypto-Channel Configuration Register (CCCR),” for more information.

14.4.4.1 Link Table Example

Figure 14-7 is an example of proper construction of link tables.

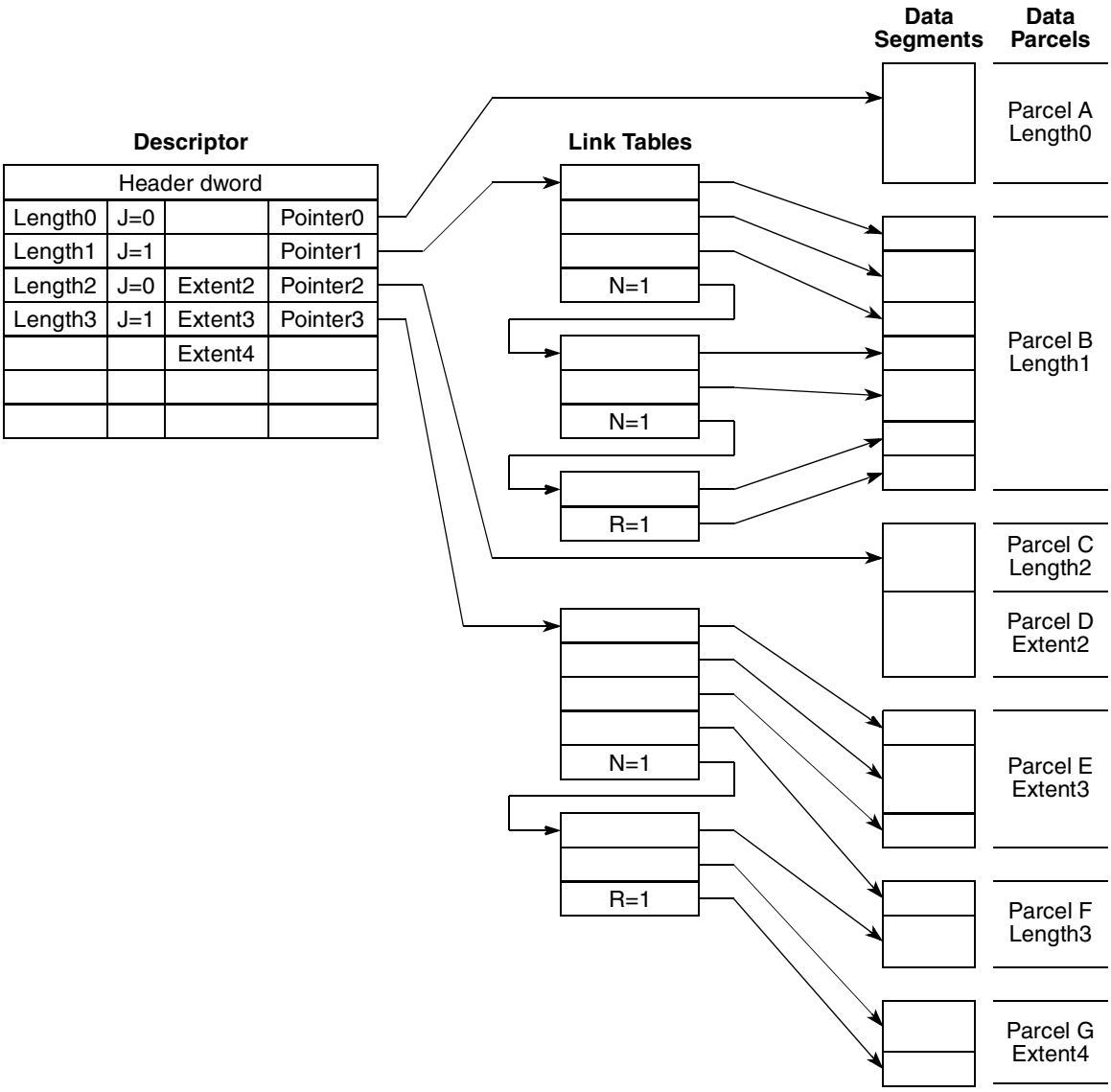


Figure 14-7. Descriptors, Link Tables, and Data Parcels

Figure 14-7 illustrates various ways that a descriptor may specify data parcels:

- The first pointer double word in the descriptor specifies Parcel A using the simplest method: the parcel is specified directly through Pointer0 and Length0.
- The next pointer double word uses a chain of link tables to specify Parcel B. Since J=1, pointer1 is used as the address of a link table. The link table specifies several ‘regular’ entries specifying data segments to be concatenated. The last word of the link table is a next entry indicating that the list continues in the ‘next’ link table. The last entry in the last link table of the chain has the R bit set.
- The last cases illustrate how one pointer in a descriptor can be used to specify multiple parcels. Pointer2 and Length2 specify Parcel C, then Parcel D follows immediately afterwards, with length

specified by Extent2. Pointer3 is used for three data parcels (E, F, and G), this time using link tables.

To demonstrate use of a link table, assume that the current descriptor type calls for the channel to access a data parcel using Pointer3 and Extent3 fields, and assume that J3=1. Due to the J3 value, Pointer3 is not used as a data address but instead used as the address of a link table. The channel begins by reading the first four long words starting at Pointer3 into an internal link table buffer.

Using the first entry of the link table, the channel starts accessing the data parcel by reading SEGLEN bytes beginning at SEGADR. If the required data parcel size (Extent3) is greater than this first SegLen, the channel moves on to the next entry of the link table, and reads SEGLEN bytes starting at SEGADR. While there are more bytes to be read in the data parcel, this process continues. If the channel's link table buffer is exhausted, the channel reads the next four long words of the link table into its link table buffer. If a link table entry is encountered in which the N bit is set, the channel uses the SEGADR field in that word to find the next link table in the chain. The last byte of the required parcel size (Extent3) must coincide with the last byte of a memory segment, or unpredictable results may occur.

Now assume that the channel accesses its next data parcel using Pointer3 again, this time with length given by Length3. In this case the channel continues to the next line of the link table, and begins reading the memory segment specified there. As before, the channel concatenates memory segments from as many link table entries as necessary to obtain the required number of bytes (Length3).

Similarly, the next data parcel is obtained by using Pointer3 yet again, this time with length given by Extent4.

Assume that for the current descriptor type, the Extent4 data parcel is the last one to be accessed through Pointer3. Then the link table entry that supplies the last memory segment for Extent4 has the R bit set, signifying that this is the last entry in the chain of link tables.

14.4.5 Descriptor Types

Table 14-9 shows how the pointer dwords should be used with the various descriptor types to load keys, context, and text data into the execution units, and how the required outputs should be unloaded. Additional explanation of the use of certain descriptor types and the meaning of the pointer dwords can be found in the *SEC 2.4 Descriptor Programmer's Guide*.

Table 14-9. Descriptor Pointer Dword Usage

Descriptor Type	Pointer DWORD 1	Pointer DWORD 2	Pointer DWORD 3	Pointer DWORD 4	EXTENT 4	Pointer DWORD 5	EXTENT 5	Pointer DWORD 6	EXTENT 6	Pointer DWORD 7
0000_0	nil	Cipher IV	Cipher Key	In FIFO	nil	Out FIFO	nil	Cipher IV Out	nil	nil
0001_0	nil	Cipher IV	Cipher Key	In FIFO	nil	Out FIFO	nil	Cipher IV Out	nil	nil
0010_0	HMAC Key	HMAC Data	Cipher Key	Cipher IV	nil	In FIFO	nil	Out FIFO	nil	HMAC Out
0011_0	Reserved									
0100_0	Reserved									
0101_0	nil	ARC-4 Context (In FIFO)	ARC-4 Key	In FIFO	nil	Out FIFO	nil	ARC-4 Context (Out FIFO)	nil	nil
0110_ 0	Reserved									
0111_0	Reserved									
1000_0	N	B	A	E	nil	B Out	nil	nil	nil	nil
1001_0	Reserved									
1010_0	Reserved									
1011_ 0	Reserved									
1100_0	HMAC Key	HMAC Data	AES Key	AES Ctx	nil	In FIFO	nil	Out FIFO	nil	HMAC Out
1101_0	Reserved									
1110_0	Reserved									
1111_0	Reserved									
0000_1	HMAC Key	HMAC Data	Cipher IV	Cipher Key	nil	In FIFO	nil	Out FIFO	HMAC Out	Cipher IV Out
0001_1	nil	AES-Ctx	AES Key	In FIFO	nil	In FIFO	nil	Out FIFO	nil	AES-Ctx-Out
0010_1	HMAC Key	AES-Ctx	AES Key	In FIFO	In FIFO	Out FIFO	In FIFO	HMAC Out	nil	AES-Ctx-Out
0011_1	A0	A1	A2	A3	nil	B0	nil	B1	nil	'Build'
0100_1	N	E	'Build'	B1 Out	nil	B2 Out	nil	B3 Out	nil	nil
0101_1	N	'Build'	B2	B3	nil	B1 Out	nil	B2 Out	nil	B3 Out
1000_1 Outbound	MAC Key	Cipher IV	Cipher Key	In FIFO	In FIFO	In FIFO	MAC In	Out FIFO	nil	Cipher IV Out
1000_1 Inbound	MAC Key	Cipher IV	Cipher Key	nil	In FIFO	In FIFO	MAC In	Out FIFO	MAC Out	Cipher IV Out
1001_1 Outbound	MAC Key	Cipher IV	Cipher Key	In FIFO	In FIFO	In FIFO	MAC Out	Out FIFO	nil	Cipher IV Out
1001_1 Inbound	MAC Key	Cipher IV	Cipher Key	nil	In FIFO	In FIFO	MAC In	Out FIFO	MAC Out	Cipher IV Out
1010_1	nil	nil	nil	In 1	undefined	In 2	undefined	In 3	undefined	Out
others	Reserved									

14.5 Execution Units

Execution unit (EU) is the term used for a functional block that performs the mathematical permutations required by protocols used in cryptographic processing. The EUs are compatible with IPSec, IKE, SSL/TLS, iSCSI, SRTP, and 802.11i processing, and can work together to perform high level cryptographic tasks.

The following execution units are used in the SEC:

- Public key execution unit (PKEU)
- Data encryption standard execution unit (DEU)
- Advanced encryption standard execution unit (AESU) implementing the Rijndael symmetric key cipher.
- ARC four execution unit (AFEU)
- Message digest execution unit (MDEU)
- One private on-chip random number generator (RNG)

Working together, the EUs can perform high-level cryptographic tasks, such as IPSec Encapsulating Security Protocol (ESP) and digital signature. The remainder of this chapter provides details about the execution units themselves.

14.5.1 Public Key Execution Unit (PKEU)

This section contains details about the public key execution unit (PKEU), including detailed register map, modes of operation, status and control registers, and the parameter RAMs. The following sections describe PKEU registers and parameter memories.

14.5.1.1 PKEU Mode Register (PKEUMR)

PKEUMR specifies the internal PKEU routine to be executed. The mode register is cleared when the PKEU is reset or re-initialized. Setting a reserved mode bit generates a data error. If the mode register is modified during processing, a context error will be generated.

Figure 14-8 shows the PKEU Mode Register, and Table 14-10 lists the possible mode field values.

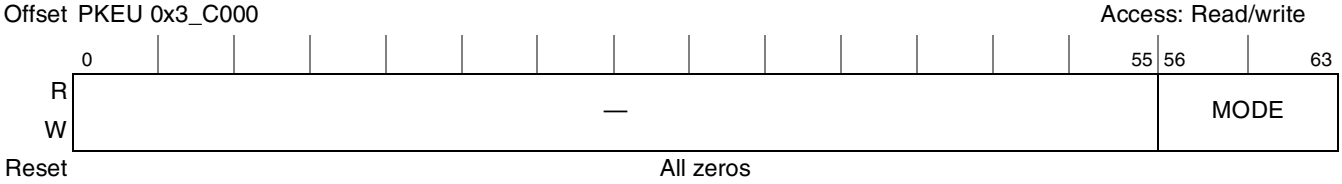


Figure 14-8. PKEU Mode Register

Table 14-10 lists the possible mode field values. Depending on the value written to PKEU[Mode] field depends on the routine used. Parameter memories are referred to for the base address, as shown.

Table 14-10. Mode Field Description

Routine Name	Routine Description	Mode [56–63]
RESERVED	Reserved	0x00
CLEARMEMORY	Clear memory	0x01
MOD_EXP	FP: Exponentiate mod N and deconvert from Montgomery format	0x02
MOD_R2MODN	FP: Compute Montgomery converter (R2 mod N)	0x03
MOD_RRMODP	FP: Compute Montgomery converter for Chinese Remainder Theorem (RnRp mod N)	0x04
EC_FP_AFF_PTMULT	FP EC: Multiply key times point in affine coordinates	0x05
EC_F2M_AFF_PTMULT	F2m EC: Multiply key times point in affine coordinates	0x06
EC_FP_PROJ_PTMULT	FP EC: Multiply key times point in projective coordinates	0x07
EC_F2M_PROJ_PTMULT	F2m EC: Multiply key times point in projective coordinates	0x08
EC_FP_ADD	FP EC: Add two points in projective coordinates	0x09
EC_FP_DOUBLE	FP EC: Double a point in projective coordinates	0x0A
EC_F2M_ADD	F2m EC: Add two points in projective coordinates	0x0B
EC_F2M_DOUBLE	F2m EC: Double a point in projective coordinates	0x0C
F2M_R2	F2m: Compute Montgomery converter (R2 mod N)	0x0D
F2M_INV[1]	F2m: Invert mod N	0x0E
MOD_INV[2]	FP: Invert mod N	0x0F
MOD_ADD	FP: Add mod N	0x10
MOD_SUB	FP: Subtract mod N	0x20
MOD_MULT1_MONT	FP: Multiply mod N in Montgomery format	0x30
MOD_MULT2_DECONV	FP: Multiply mod N and deconvert from Montgomery format	0x40
F2M_ADD	F2m: Add mod N	0x50
F2M_MULT1_MONT	F2m: Multiply mod N in Montgomery format	0x60
F2M_MULT2_DECONV	F2m: Multiply mod N and deconvert from Montgomery format	0x70
RSA_SSTEP	FP: Exponentiate mod N (combines MOD_R2MODN, F2M_MULT1_MONT, and MOD_EXP)	0x80
SPK_BUILD	Build PK data structure	0xFF

1 F2M_INV returns incorrect results for modulus sizes greater than 480 bits.

2 MOD_INV returns incorrect results for modulus sizes greater than 480 bits.

14.5.1.2 PKEU Key Size Register (PKEUKSR)

The PKEU key size register, shown in Figure 14-9, reflects the number of significant bytes to be used from PKEU parameter memory E in performing modular exponentiation or elliptic curve point multiplication. Note that leading zeros are note significant and are not considered part of the key (modulus) size. The range of values for this register, when performing either modular exponentiation or elliptic curve point multiplication, is from 1 to 256. Specifying a key size outside of this range will cause a key size error in the PKEU interrupt status register (PKEUISR[KSE] is set).

NOTE

Writing to this register while in debug mode generates an illegal size error. Disable the illegal size error in the associated Interrupt Control Register prior to performing debug operations.

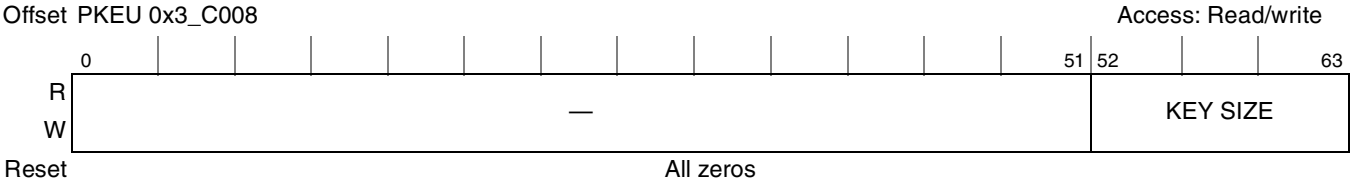


Figure 14-9. PKEU Key Size Register

14.5.1.3 PKEU AB Size Register (PKEUABS)

PKEU ABS (Figure 14-10) represents the operand size for the specific operands whenever it is required. The unit of the value written into PKEUABS[AB Size] is in bits, even though internally the PKEU imposes a 32-bit alignment. Any data beyond the number of bits in PKEUABS, either in A- and B-ram (operands) will be ignored. No error checking is performed whether the operand sizes are greater than the prime modulus or the field size and this may cause a wrong result. In other words, it is assumed that operands are modulo reduced before being written into the PKEU. Hence, PKEUABS[AB Size] must be less than or equal to data size for a correct result. If PKEUABS is modified during processing, an error will be generated.

An illegal data size error is generated as follows:

- For all non-ECC routines, a data size > 256 generates an illegal data size error.
- For all ECC routines, a data size > 64 generates an illegal data size error.

Setting PKEUABS[AB Size] = 0 (either intentionally or by ignoring it and not writing it at all) generates an illegal size error, except for routines such as the CLEAR_MEM routine that do not require an A or B operand.

NOTE

Writing to this register while in debug mode generates an illegal size error. Disable the illegal size error in the associated interrupt control register before performing debug operations.

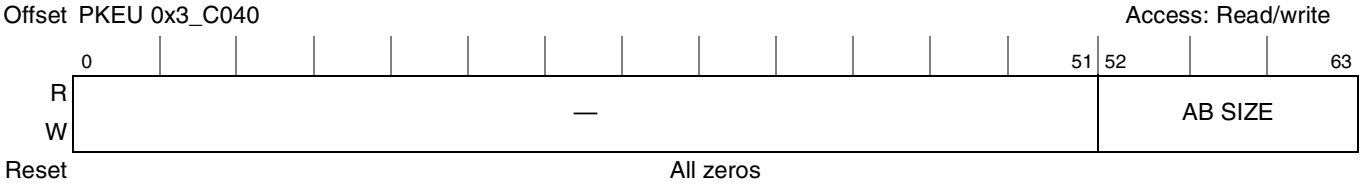


Figure 14-10. PKEU AB Size Register

14.5.1.4 PKEU Data Size Register (PKEUDSR)

PKEUDSR, shown in Figure 14-11, specifies the size in bits of the significant portion of the modulus or irreducible polynomial. Any value written to this register that is a multiple of 32 bits (for example, 128 bits or 160 bits) will be represented internally as the same value (128 bits or 160 bits). Any value written that is not a multiple of 32 bits (for example, 132 bits or 161 bits) will be represented internally as the next larger 32 bit multiple (160 bits or 196 bits). This internal rounding up to the next 32-bit multiple is described for information only. The minimum size valid for all routines to operate properly is 97 bits (internally 128 bits). The maximum size to operate properly is 2048 bits. A value in bits larger than 2048 results in a data size error.

NOTE

Writing to this register while in debug mode generates an illegal size error. Disable the illegal size error in the associated interrupt control register prior to performing debug operations.

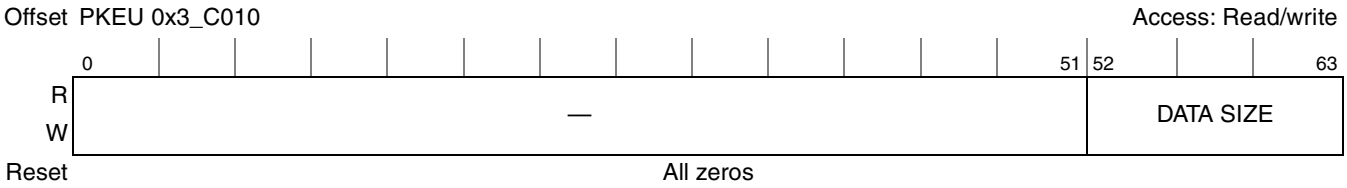


Figure 14-11. PKEU Data Size Register (PKEUDSR)

14.5.1.5 PKEU Reset Control Register (PKEURCR)

PKEURCR, shown in Figure 14-12, contains three reset options specific to the PKEU.

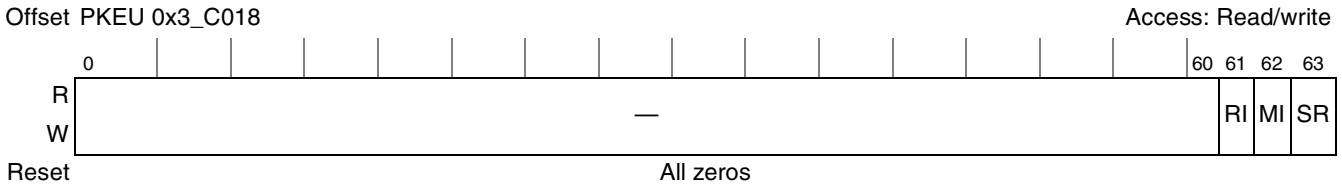


Figure 14-12. PKEU Reset Control Register (PKEURCR)

Table 14-11 describes PKEURCR fields.

Table 14-11. PKEURCR Field Descriptions

Bits	Name	Description
0–60	—	Reserved
61	RI	Reset interrupt. Writing this bit active high causes PKEU interrupts signaling DONE and ERROR to be reset. It further resets the state of the PKEU interrupt status register. 0 Do not reset interrupt logic. 1 Reset interrupt logic.
62	MI	Module initialization. Module initialization is nearly the same as software reset, except that the interrupt control register remains unchanged. This module initialization includes execution of an initialization routine, completion of which is indicated by the RD (reset done) bit in the PKEU status register (Section 14.5.1.6, “PKEU Status Register (PKEUSR)”). 0 Do not reset most of PKEU. 1 Reset most of PKEU.
63	SR	SW reset. Software reset is functionally equivalent to hardware reset (the RESET# pin), but only for the PKEU. All registers and internal state are returned to their defined reset state. Upon negation of SW_RESET, the PKEU will enter a routine to perform proper initialization of the parameter memories. The RD (reset done) bit in the PKEU status register will indicate when this initialization routine is complete. (See Section 14.5.1.6, “PKEU Status Register (PKEUSR).”) 0 Do not reset full PKEU. 1 Full PKEU reset.

14.5.1.6 PKEU Status Register (PKEUSR)

PKEUSR fields reflect the state of PKEU internal fields. PKEUSR is read only. Writing to it results in an address error being reflected in the PKEU interrupt status register (PKEUISR[AE] is set).

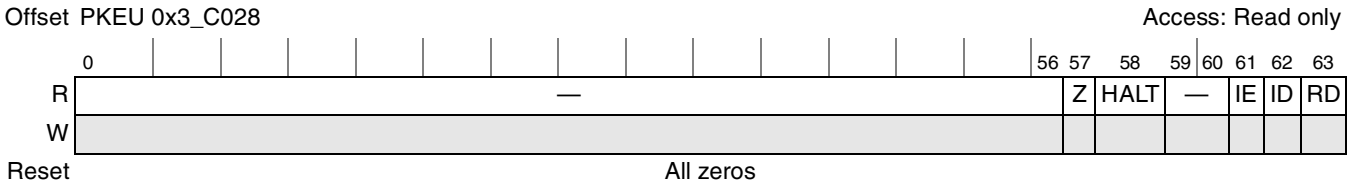


Figure 14-13. PKEU Status Register (PKEUSR)

Table 14-12 describes PKEUSR fields.

Table 14-12. PKEU Status Register Field Descriptions

Bits	Name	Description
0–56	—	Reserved
57	Z	Zero. Reflects the state of the PKEU zero detect bit when last sampled. Only particular instructions within routines cause zero to be modified, so this bit should be used with great care.
58	HALT	Halt indicates whether the PKEU has halted due to an error. 0 PKEU not halted 1 PKEU halted **Note:** Because the error causing the PKEU to stop operating may be masked to the interrupt status register, the status register is used to provide a second source of information regarding errors preventing normal operation.

Table 14-12. PKEU Status Register Field Descriptions (continued)

Bits	Name	Description
59–60	—	Reserved
61	IE	Interrupt error. Reflects the state of the ERROR interrupt signal, as sampled by the controller interrupt status register (see Section 14.7.2.2, “Interrupt Status Register (ISR)”). 0 PKEU is not signaling error 1 PKEU is signaling error
62	ID	Interrupt done. Reflects the state of the DONE interrupt signal, as sampled by the controller interrupt status register (see Section 14.7.2.2, “Interrupt Status Register (ISR)”). 0 PKEU is not signaling done 1 KEU is signaling done
63	RD	Reset done. 0 Reset in progress 1 Reset done. PKEU has completed its internal reset sequence.

14.5.1.7 PKEU Interrupt Status Register (PKEUISR)

The PKEU interrupt status register tracks the state of possible errors, if those errors are not masked, through the PKEU interrupt control register (PKEUICR). The definition of each bit in the PKEUISR is shown in Figure 14-14.

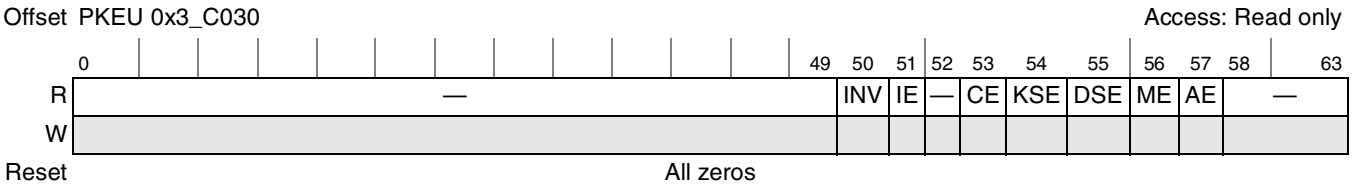


Figure 14-14. PKEU Interrupt Status Register (PKEUISR)

Table 14-13 describes PKEUISR fields.

Table 14-13. PKEUISR Field Descriptions

Bits	Name	Description
0–49	—	Reserved
50	INV	Inversion error. Indicates that the inversion routine has a zero operand. 0 No inversion error detected 1 Inversion error detected
51	IE	Internal error. An internal processing error was detected while the PKEU was operating. 0 No error detected 1 Internal error **Note:** This bit will be asserted any time an enabled error condition occurs and can only be cleared by setting the corresponding bit in the Interrupt Control Register or by resetting the PKEU.
52	—	Reserved
53	CE	Context error. A PKEU key register, the key size register, the data size register, or mode register was modified while the PKEU was operating. 0 No error detected 1 Context error

Table 14-13. PKEUISR Field Descriptions (continued)

Bits	Name	Description
54	KSE	Key size error. Value outside the bounds of 1–256 bytes was written to the PKEU key size register 0 No error detected 1 Key size error detected
55	DSE	Data size error. Value outside the bounds 97– 2048 bits was written to the PKEU data size register 0 No error detected 1 Data size error detected
56	ME	Mode error. An illegal value was detected in the mode register. 0 No error detected 1 Mode error **Note:** Writing to reserved bits in a mode register is a likely source of error.
57	AE	Address error. Illegal read or write address was detected within the PKEU address space. 0 No error detected 1 Address error
58–63	—	Reserved

14.5.1.8 PKEU Interrupt Control Register (PKEUICR)

The PKEU interrupt control register controls the result of detected errors. For a given error (as defined in Section 14.5.1.7, “PKEU Interrupt Status Register (PKEUISR)”), if the corresponding bit in this register is set, then the error is disabled; no error interrupt occurs and the interrupt status register is not updated to reflect the error. If the corresponding bit is not set, then upon detection of an error, PKEUISR is updated to reflect the error, causing assertion of the error interrupt signal, and causing the module to halt processing.

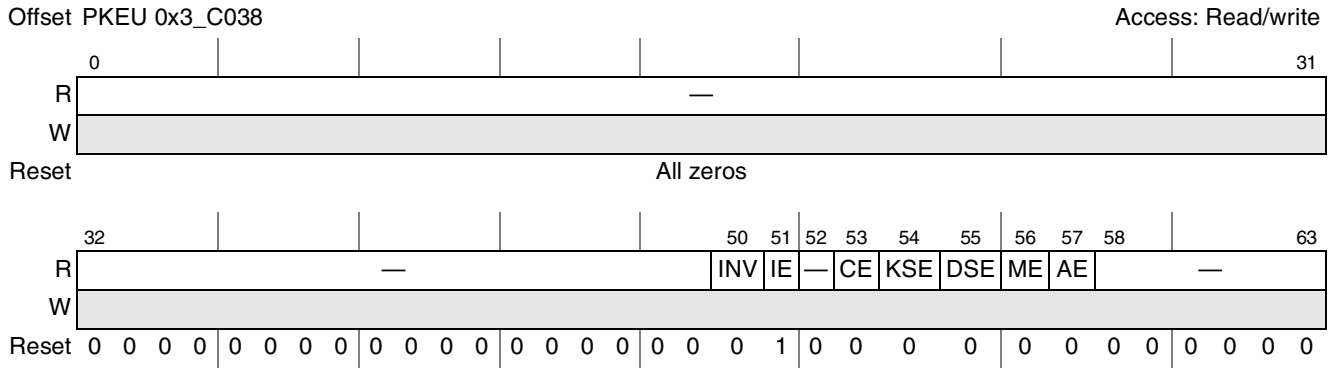


Figure 14-15. PKEU Interrupt Control Register (PKEUICR)

Table 14-14 describes PKEUICR fields.

Table 14-14. PKEUICR Field Descriptions

Bits	Name	Description
0–49	—	Reserved
50	INV	Inversion error 0 Inversion error enabled 1 Inversion error disabled
51	IE	Internal error 0 Internal error enabled 1 Internal error disabled
52	—	Reserved
53	CE	Context error 0 Context error enabled 1 Context error disabled
54	KSE	Key size error 0 Key size error enabled 1 Key size error disabled
55	DSE	Data size error 0 Data size error enabled 1 Data size error disabled
56	ME	Mode error 0 Mode error enabled 1 Mode error disabled
57	AE	Address error 0 Address error enabled 1 Address error disabled
58–63	—	Reserved

14.5.1.9 PKEU EU-Go Register (PKEUEUG)

The EU-Go register in the PKEU is used to indicate the start of a new computation. Writing to this register causes the PKEU to execute the function requested by the mode register, per the contents of the parameter memories listed below. Note that this register has no data size, and during the write operation, the host data bus is not read. Hence, any data value is accepted. Normally, a write operation with a zero data value is performed. Moreover, no read operation from this register is meaningful, but no error is generated, and a zero value is always returned. PKEUEUG is only used when the is operated as a slave. The descriptors and crypto-channel activate the PKEU (through an internally generated write to PKEUEUG) when the acts as an initiator.

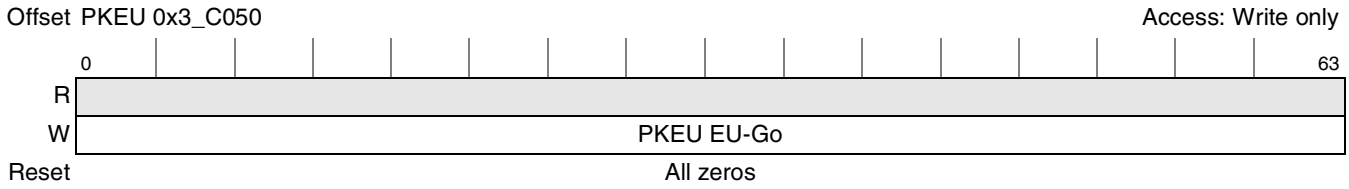


Figure 14-16. PKEU EU-Go Register (PKEUEUG)

14.5.1.10 PKEU Parameter Memories

The PKEU uses four 2048-bit memories to receive and store operands for the arithmetic operations the PKEU will be asked to perform. In addition, results are stored in one particular parameter memory.

All these memories store data in the same format: least significant data byte in the least significantly addressed byte, both data significance and addressing significance increasing identically and simultaneously.

14.5.1.10.1 PKEU Parameter Memory A

This 2048-bit memory is typically used as an input parameter memory space. For modular arithmetic routines, this memory operates as one of the operands of the desired function. For elliptic curve routines, this memory is segmented into four 512-bit memories, and is used to specify particular curve parameters and input values.

14.5.1.10.2 PKEU Parameter Memory B

This 2048-bit memory is typically used as an input parameter memory space, as well as the result memory space. For modular arithmetic routines, this memory serves as one of the operands of the desired function, as well as the result memory space. For elliptic curve routines, this memory is segmented in to four 512-bit memories, and is used to specify particular curve parameters and input values, as well as to store result values.

14.5.1.10.3 PKEU Parameter Memory E

This 2048-bit memory is non-segmentable, and stores the exponent for modular exponentiation, or the multiplier k for elliptic curve point multiplication. This memory space is write only; a read of this memory space will cause address error to be reflected in the PKEUISR.

14.5.1.10.4 PKEU Parameter Memory N

This 2048-bit memory is non-segmentable, and stores the modulus for modular arithmetic and F_p elliptic curve routines. For F_2M elliptic curve routines, this memory stores the irreducible polynomial.

14.5.2 Data Encryption Standard Execution Unit (DEU)

This section contains details about the data encryption standard execution unit (DEU), including detailed register map, modes of operation, status and control registers, and FIFOs.

The registers used in the DEU are documented primarily for debug and slave mode operations. If the SEC requires the use of the DEU when acting as an initiator, accessing these registers directly is unnecessary. The device drivers and on-chip controller will abstract register-level access from the user.

14.5.2.1 DEU Mode Register (DEUMR)

The DEU mode register contains 3 bits that are used to program the DEU. It also reflects the value of burst size, which is loaded by the crypto-channel during normal operation with the SEC as an initiator. Burst size is not relevant to slave mode operations, where an external host pushes and pulls data from the execution units.

DEUMR is cleared when the DEU is reset or re-initialized. Setting a reserved mode bit generates a data error. If the mode register is modified during processing, a context error will be generated.

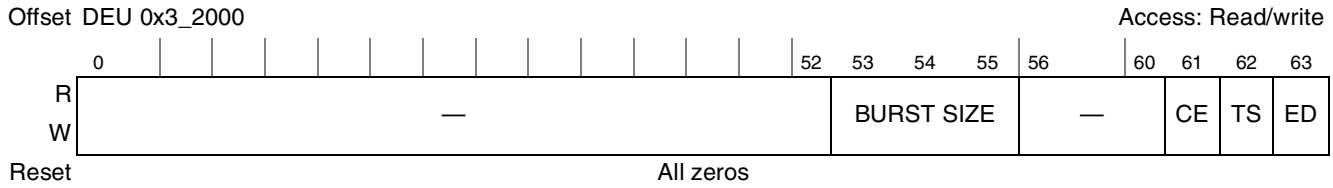


Figure 14-17. DEU Mode Register (DEUMR)

Table 14-15 describes DEU mode register fields.

Table 14-15. DEUMR Field Descriptions

Bits	Name	Description
0–52	—	Reserved
53–55	BURST SIZE	Implements flow control to allow larger than FIFO-sized blocks of data to be processed with a single key/IV. The DEU signals to the channel that a BURST SIZE amount of data is available to be pushed to or pulled from the FIFO. **Note:** This field is included to avoid confusing a user who may read this register in debug mode. Burst size should not be written directly to the DEU.
56–60	—	Reserved
61	CE	CBC/ECB. If set, DEU operates in cipher-block-chaining mode. If not set, DEU operates in electronic codebook mode. 0 ECB mode 1 CBC mode
62	TS	Triple/Single DES. If set, DEU performs the triple DES algorithm; if not set, DEU performs the single DES algorithm. 0 Perform single DES. 1 Perform triple DES.
63	ED	Encrypt/Decrypt. If set, DEU performs the encryption algorithm; if not set, DEU performs the decryption algorithm. 0 Perform decryption. 1 Perform encryption.

14.5.2.2 DEU Key Size Register (DEUKSR)

The DEUKSR indicates the number of bytes of key memory used in encrypting or decrypting. If DEUMR is set for single DES, any value other than 8 bytes automatically generates a key size error in the DEUISR. If the mode bit is set for triple DES, any value other than 16 bytes (112 bits for 2-key triple DES (K1=K3) or 24 bytes (168 bits for 3-key triple DES) generates an error. Triple DES always uses K1 to encrypt, K2 to decrypt, K3 to encrypt.

NOTE

Writing to this register while in debug mode generates an illegal size error. Disable the illegal size error in the associated Interrupt Control Register prior to performing debug operations.

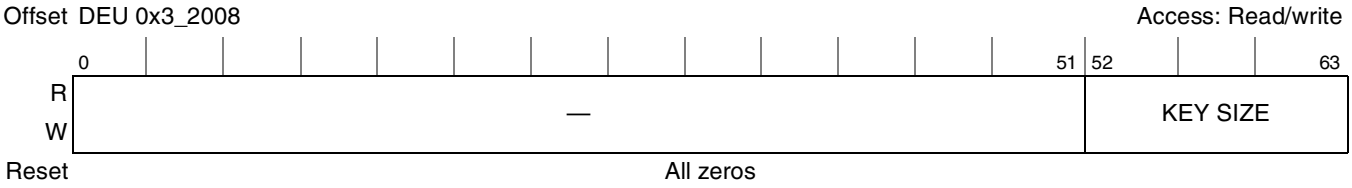


Figure 14-18. DEU Key Size Register (DEUKSR)

Table 14-16 shows the legal values for DEU key size.

Table 14-16. DEUKSR Field Descriptions

Bits	Name	Description
0–51	—	Reserved
52–63	Key Size	8 bytes = 0x08 (only legal value if mode is single DES.) 16 bytes= 0x10 (for 2 key 3DES, K1 = K3) 24 bytes= 0x18 (for 3 key 3DES)

14.5.2.3 DEU Data Size Register (DEUDSR)

The DEUDSR, shown in Figure 14-19, is used to verify that the data to be processed by the DEU is divisible by the DES algorithm block size of 64 bits. The DEU does not automatically pad messages out to 64-bit blocks; therefore, any message processed by the DEU must be divisible by 64 bits or a data size error will occur.

In normal operation, the full message length (data size) to be encrypted or decrypted by the DEU is copied from the descriptor to DEUDSR; however, only bits 58–63 are checked to determine if there is a data size error. If bits 58–63 are all zeros, the message is evenly divisible into 64-bit blocks. In target mode, the user must write the data size to DEUDSR. If the data size written is not divisible by 64-bits (bits 58–63 nonzero), a data size error will occur.

NOTE

Writing to this register while in debug mode generates an illegal size error. Disable the illegal size error in the associated Interrupt Control Register prior to performing debug operations.

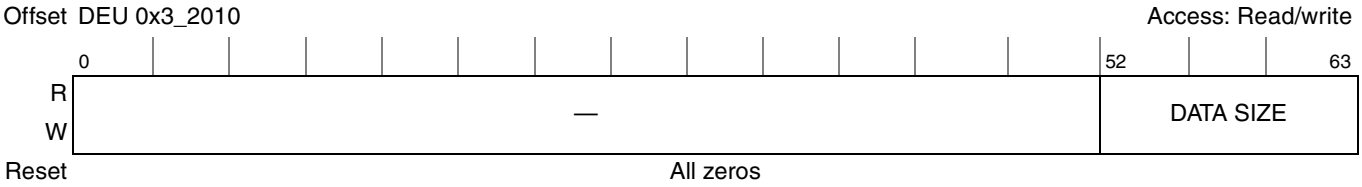


Figure 14-19. DEU Data Size Register (DEUDSR)

14.5.2.4 DEU Reset Control Register (DEURCR)

The DEURCR, shown in Figure 14-20, allows three levels reset of just DEU, as defined by the three self-clearing bits:

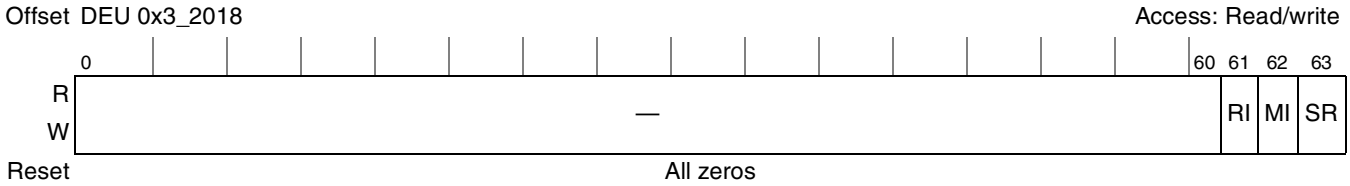


Figure 14-20. DEU Reset Control Register (DEURCR)

Table 14-17 describes DEURCR fields.

Table 14-17. DEURCR Field Descriptions

Bits	Names	Description
0–60	—	Reserved
61	RI	Reset interrupt. Writing this bit active high causes DEU interrupts signaling DONE and ERROR to be reset. It further resets the state of DEUISR. 0 Don't reset. 1 Reset interrupt logic.
62	MI	Module initialization is nearly the same as software reset, except that the interrupt control register remains unchanged. this module initialization includes execution of an initialization routine, completion of which is indicated by the RESET_DONE bit in DEUSR. 0 Don't reset. 1 Reset most of DEU.
63	SR	Software reset is functionally equivalent to hardware reset (the RESET# pin), but only for DEU. All registers and internal state are returned to their defined reset state. Upon negation of SW_RESET, the DEU will enter a routine to perform proper initialization of the parameter memories. The RESET_DONE bit in DEUSR will indicate when this initialization routine is complete 0 Don't reset. 1 Full DEU reset

14.5.2.5 DEU Status Register (DEUSR)

The DEU status register, shown in Figure 14-21, contains six fields which reflect the state of DEU internal signals. DEUSR is read only; writing to DEUSR causes an address error being reflected in DEUISR.

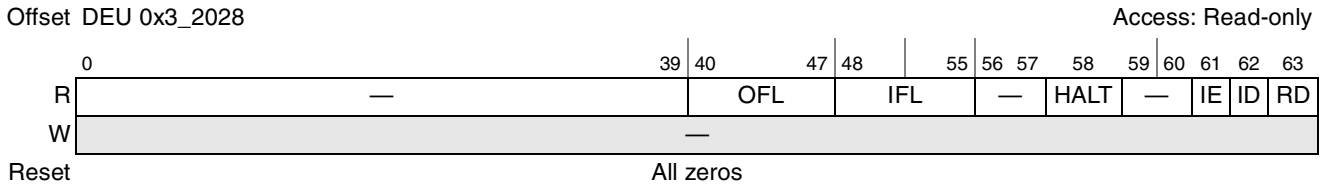


Figure 14-21. DEU Status Register (DEUSR)

Table 14-12 describes DEUSR fields.

Table 14-18. DEUSR Field Descriptions

Bits	Name	Description
0–39	—	Reserved
40–47	OFL	The number of dwords currently in the output FIFO
48–55	IFL	The number of dwords currently in the input FIFO
56–57	—	Reserved
58	HALT	Halt. Indicates that the DEU has halted due to an error. 0 DEU not halted 1 DEU halted **Note:** Because the error causing the DEU to stop operating may be masked to the interrupt status register, the status register is used to provide a second source of information regarding errors preventing normal operation.
59–60	—	Reserved
61	IE	Interrupt error. Reflects the state of the ERROR interrupt signal, as sampled by the controller interrupt status register (Section 14.7.2.2, “Interrupt Status Register (ISR)”). 0 DEU is not signaling error 1 DEU is signaling error
62	ID	Interrupt done. Reflects the state of the DONE interrupt signal, as sampled by the controller interrupt status register (Section 14.7.2.2, “Interrupt Status Register (ISR)”). 0 DEU is not signaling done 1 DEU is signaling done
63	RD	Reset done. 0 Reset in progress 1 Reset done. DEU has completed its internal reset sequence.

14.5.2.6 DEU Interrupt Status Register (DEUISR)

The DEU interrupt status register, shown in Figure 14-22, tracks the state of possible errors, if those errors are not masked, via the DEU interrupt control register.

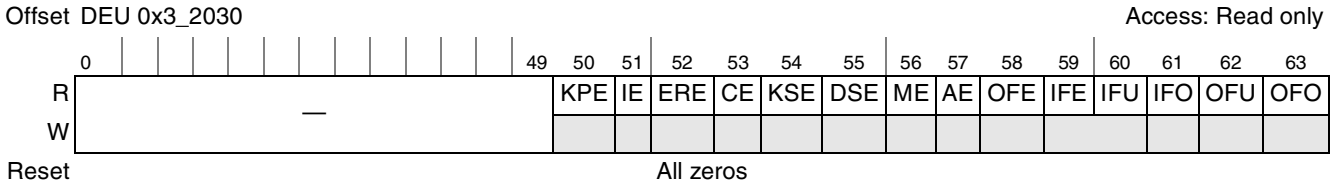


Figure 14-22. DEU Interrupt Status Register (DEUISR)

Table 14-19 describes DEUISR fields.

Table 14-19. DEUISR Field Descriptions

Bits	Name	Description
0–49	—	Reserved
50	KPE	Key parity error. Defined parity bits in the keys written to the key registers did not reflect odd parity correctly. (Note that key register 2 and key register 3 are checked for parity only if the appropriate DEU mode register bit indicates triple DES. Also, key register 3 is checked only if key size reg = 24. Key register 2 is checked only if key size reg = 16 or 24.) 0 No error detected 1 Key parity error
51	IE	Internal error. An internal processing error was detected while performing encryption. 0 No error detected 1 Internal error **Note:** IE is asserted anytime an enabled error condition occurs and can only be cleared by setting the corresponding bit in the DEUICR or by resetting the DEU.
52	ERE	Early read error. The DEU IV register was read while the DEU was performing encryption. 0 No error detected 1 Early read error
53	CE	Context error. A DEU Key Register, the Key Size Register, Data Size Register, Mode Register, or IV Register was modified while DEU was performing encryption. 0 No error detected 1 Context error
54	KSE	Key size error. An inappropriate value (8 being appropriate for single DES, and 16 and 24 being appropriate for triple DES) was written to the DEU key size register 0 No error detected 1 Key size error
55	DSE	Data size error: A value was written to the DEU Data Size Register that is not a multiple of 64 bits. 0 No error detected 1 Data size error
56	ME	Mode error. An illegal value was detected in the mode register. Note: writing to reserved bits in mode register is likely source of error. 0 No error detected 1 Mode error
57	AE	Address error. An illegal read or write address was detected within the DEU address space. 0 No error detected 1 Address error
58	OFE	Output FIFO error. The DEU output FIFO was detected non-empty upon write of DEU data size register. 0 No error detected 1 Output FIFO non-empty error
59	IFE	Input FIFO error. The DEU input FIFO was detected non-empty upon generation of DONE interrupt. 0 No error detected 1 Input FIFO non-empty error
60	IFU	Input FIFO underflow. The DEU input FIFO has been read while empty. 0 No error detected 1 Input FIFO has had underflow error

Table 14-19. DEUISR Field Descriptions (continued)

Bits	Name	Description
61	IFO	Input FIFO overflow. The DEU input FIFO has been pushed while full. 0 No error detected 1 Input FIFO has overflowed **Note:** When operating as a master, the implements flow-control, and FIFO size is not a limit to data input. When operated as a target, the cannot accept FIFO inputs larger than 512 bytes without overflowing.
62	OFU	Output FIFO underflow. The DEU output FIFO has been read while empty. 0 No error detected 1 Output FIFO has underflow error
63	OFO	Output FIFO overflow. The DEU output FIFO has been pushed while full. 0 No error detected 1 Output FIFO has overflowed

14.5.2.7 DEU Interrupt Control Register (DEUICR)

The DEU interrupt control register controls the result of detected errors. For a given error (as defined in Section 14.5.2.6, “DEU Interrupt Status Register (DEUISR)”),if the corresponding bit in this register is set, then the error is ignored, no error interrupt occurs and DEUISR is not updated to reflect the error. If the corresponding bit is not set, then upon detection of an error, DEUISR is updated to reflect the error, causing assertion of the error interrupt signal, and causing the module to halt processing.

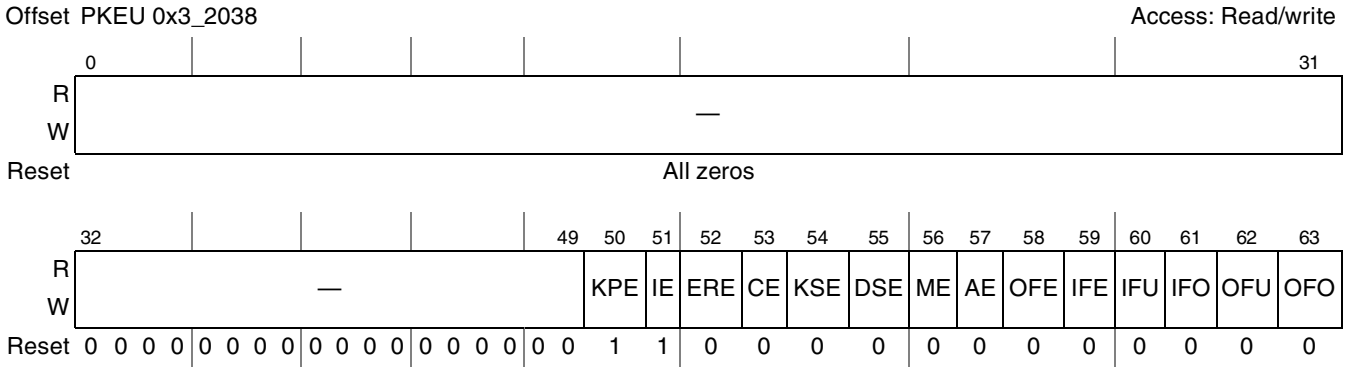


Figure 14-23. DEU Interrupt Control Register (DEUICR)

Table 14-20. DEUICR Field Descriptions

Bits	Name	Description
0–49	—	Reserved
50	KPE	Key parity error. The defined parity bits in the keys written to the key registers did not reflect odd parity correctly. (Note that key register 2 and key register 3 are only checked for parity if the appropriate DEU mode register bit indicates triple DES. 0 Key parity enabled 1 Key parity error disabled
51	IE	Internal error. An internal processing error was detected while performing encryption. 0 Internal error enabled 1 Internal error disabled

Table 14-20. DEUICR Field Descriptions (continued)

Bits	Name	Description
52	ERE	Early read error. The DEU IV Register was read while the DEU was performing encryption. 0 Early read error enabled 1 Early read error disabled
53	CE	Context error. A DEU key register, the key size register, the data size register, the mode register, or IV register was modified while DEU was performing encryption. 0 Context error enabled 1 Context error disabled
54	KSE	Key size error. An inappropriate value (8 being appropriate for single DES, and 16 and 24 being appropriate for Triple DES) was written to the DEU key size register 0 Key size error enabled 1 Key size error disabled
55	DSE	Data size error (DSE): A value was written to the DEU data size register that is not a multiple of 8 bytes. 0 Data size error enabled 1 Data size error disabled
56	ME	Mode error. An illegal value was detected in the mode register. 0 Mode error enabled 1 Mode error disabled
57	AE	Address error. An illegal read or write address was detected within the DEU address space. 0 Address error enabled 1 Address error disabled
58	OFE	Output FIFO error. The DEU output FIFO was detected non-empty upon write of DEU data size register 0 Output FIFO non-empty error enabled 1 Output FIFO non-empty error disabled
59	IFE	Input FIFO error. The DEU input FIFO was detected non-empty upon generation of DONE interrupt 0 Input FIFO non-empty error enabled 1 Input FIFO non-empty error disabled
60	IFU	Input FIFO underflow. The DEU input FIFO has been read while empty. 0 Input FIFO Underflow error enabled 1 Input FIFO Underflow error disabled
61	IFO	Input FIFO overflow. The DEU input FIFO has been pushed while full. 0 Input FIFO overflow error enabled 1 Input FIFO overflow error disabled **Note:** When operating as a master, the implements flow-control, and FIFO size is not a limit to data input. When operated as a target, the cannot accept FIFO inputs larger than 512 bytes without overflowing.
62	OFU	Output FIFO underflow. The DEU output FIFO has been read while empty. 0 Output FIFO underflow error enabled 1 Output FIFO underflow error disabled
63	OFO	Output FIFO overflow. The DEU output FIFO has been pushed while full. 0 Output FIFO Overflow error enabled 1 Output FIFO Overflow error disabled

14.5.2.8 DEU EU-Go Register (DEUEUG)

The EU-Go register in the DEU is used to indicate a DES operation may be completed. After the final message block is written to the input FIFO, DEUEUG must be written. The value in the data size register

will be used to determine how many bits of the final message block (always 64) will be processed. Note that this register has no data size, and during the write operation, the host data bus is not read. Hence, any data value is accepted. Normally, a write operation with a zero data value is performed. Moreover, no read operation from this register is meaningful, but no error is generated, and a zero value is always returned. Writing to DEUEUG is merely a trigger causing the DEU to process the final block of a message, allowing it to signal DONE.

DEUEUG is only used when the SEC is operated as a slave. The descriptors and crypto-channel activate the DEU (through an internally generated write to DEUEUG) when the SEC acts as an initiator.

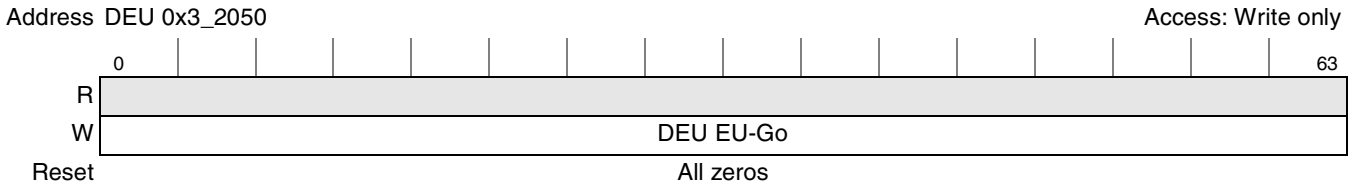


Figure 14-24. DEU EU-Go Register (DEUEUG)

14.5.2.9 DEU IV Register (DEUIV)

For CBC mode, the initialization vector is written to and read from DEUIV. The value of this register changes as a result of the encryption process and reflects the context of DEU. Reading DEUIV while the module is processing data generates an error interrupt.

14.5.2.10 DEU Key Registers (DEUK1–DEUK3)

The DEU uses three write-only key registers to perform encryption and decryption. In Single DES mode, only DEUK1 may be written. The value written to DEUK1 is simultaneously written to DEUK3, auto-enabling the DEU for 112-bit triple DES if DEUKSR indicates 2-key 3DES is to be performed (key size = 16 bytes). To operate in 168-bit triple DES, DEUK1 must be written first, followed by DEUK2, then DEUK3.

Reading any of these memory locations generates an address error interrupt.

14.5.2.11 DEU FIFOs

DEU uses an input FIFO/output FIFO pair to hold data before and after the encryption process. These FIFOs are multiply addressable, but those multiple addresses point only to the appropriate end of the appropriate FIFO. A write to anywhere in the DEU FIFO address space causes the 64-bit-word to be pushed onto the DEU input FIFO, and a read from anywhere in the DEU FIFO address space causes a 64-bit-word to be popped off the DEU output FIFO. Overflows and underflows caused by reading or writing the DEU FIFOs are reflected in the DEU interrupt status register.

14.5.3 ARC Four Execution Unit (AFEU)

This section contains details about the ARC four execution unit (AFEU), including detailed register map, modes of operation, status and control registers, S-box memory, and FIFOs.

The registers used in the AFEU are documented primarily for debug and slave mode operations. If the SEC requires the use of the AFEU when acting as an initiator, accessing these registers directly is unnecessary. The device drivers and the on-chip controller will abstract register level access from the user.

14.5.3.1 AFEU Mode Register (AFEUMR)

Shown in Figure 14-25, the AFEU mode register contains three bits which are used to program the AFEU. It also reflects the value of burst size, which is loaded by the crypto-channel during normal operation with the SEC as an initiator. Burst size is not relevant to slave mode operations, where an external host pushes and pulls data from the execution units.

AFEUMR is cleared when the AFEU is reset or re-initialized. Setting a reserved mode bit generates a data error. If AFEUMR is modified during processing, a context error is generated.

14.5.3.2 Host-Provided Context through Prevent Permute

In the default mode of operation, the host provides the key and key size to the AFEU. The initial memory values in the S-Box are permuted with the key to create new S-Box values, which are used to encrypt the plaintext.

If AFEUMR[PP] is set (prevent permute mode is enabled), the AFEU will not require a key. Rather, the host will write the context to the AFEU, and message processing will occur using the provided context. This mode is used to resume processing of a message using the already permuted S-Box. The context may be written through the FIFO if AFEUMR[CS] is set.

14.5.3.2.1 Dump Context

This mode may be independently specified (using AFEUMR[DC]) in addition to host-provided context mode. In this mode, once message processing is complete and the output data is read, the AFEU will make the current context data available for reads through the output FIFO.

NOTE

After the initial key permute to generate a context for an AFEU encrypted session, all subsequent messages will re-use that context, such that it is loaded, modified during the encryption, and unloaded, similar to the use of a CBC initialization vector in DES operations. A new context is generated (through key permute) according to a re-keying interval specified by the security protocol. Context should never be loaded to encrypt a message if a key is loaded and permuted at the same time.

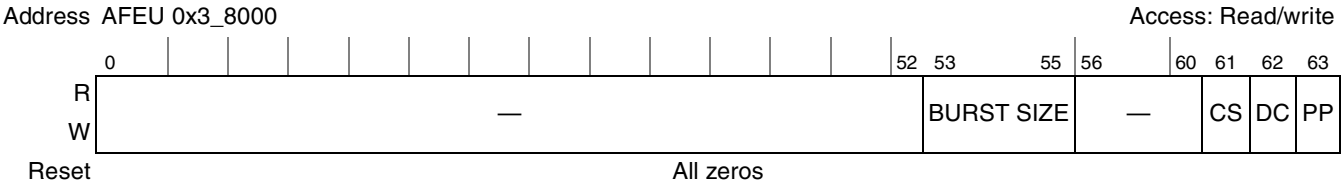


Figure 14-25. AFEU Mode Register (AFEUMR)

Table 14-21 describes AFEU mode register fields.

Table 14-21. AFEUMR Field Descriptions

Bits	Name	Description
0–52	—	Reserved
53–55	BURST SIZE	The implements flow control to allow larger than FIFO sized blocks of data to be processed with a single key/context. The AFEU signals to the channel that a ‘burst size’ amount of data is available to be pushed to or pulled from the FIFO. **Note:** The inclusion of this field in the AFEU Mode Register is to avoid confusing a user who may read this register in debug mode. Burst size should not be written directly to the AFEU.
56–60	—	Reserved
61	CS	Context source. If set, this causes the context to be moved from the input FIFO into the S-box prior to starting encryption/decryption. Otherwise, context should be directly written to the context registers. Context source is checked only if the prevent permute (PP) bit is set. 0 Context not from FIFO 1 Context from input FIFO
62	DC	Dump context. If set, this causes the context to be moved from the S-box to the output FIFO following assertion AFEU’s DONE interrupt. 0 Do not dump context 1 After cipher, dump context
63	PP	Prevent permute. Normally, AFEU receives a key and uses that information to randomize the S-box. If reusing a context from a previous descriptor, PP should be set to prevent AFEU from repeating this permutation step. 0 Perform S-box permutation 1 Do not permute

14.5.3.3 AFEU Key Size Register (AFEUKSR)

As shown in Figure 14-26, AFEUKSR indicates the number of bytes of key memory that should be used in performing S-box permutation. Any key data beyond the number of bytes in AFEUKSR is ignored. AFEUKSR is cleared when the AFEU is reset or re-initialized. If the key size specified is less than 1 or greater than 16, a key size error will be generated. If AFEUKSR is modified during processing, a context error will be generated. Note that although the AFEU supports key lengths as short as 1 byte, a 1-byte key offers little security. Most uses of ARC4 specify keys of 5–16 bytes.

NOTE

Writing to this register while in debug mode generates an illegal size error. Disable the illegal size error in the associated interrupt control register before performing debug.

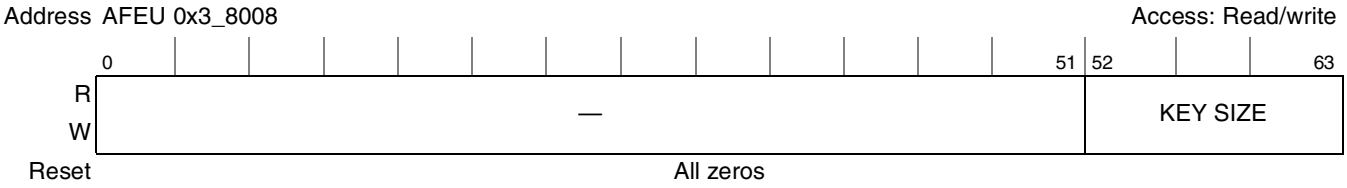


Figure 14-26. AFEU Key Size Register (AFEUKSR)

NOTE

The device driver will create properly formatted descriptors for situations requiring a key permute prior to ciphering. When operating the SEC as a slave (typically while in debug mode), the user must set the AFEU mode register (AFEUMR) to perform ‘permute with key’, then write the key data to the AFEU key registers, then write the key size to the key size register (AFEUKSR[Key Size]). The AFEU will start permuting the memory with the contents of the AFEU key registers immediately after AFEUKSR[Key Size] is written.

14.5.3.4 AFEU Context/Data Size Register (AFEUDSR)

The AFEU context/data size register, shown in Figure 14-27, stores the number of bits in the final message block. AFEUDSR is cleared when the AFEU is reset or re-initialized. The last message block can be between 8 to 64 bits. If a data size that is not a multiple of 8 bits is written, a data size error will be generated. A data size of 0 is illegal and results in the associated crypto-channel locking, requiring a crypto-channel and AFEU reset.

AFEUDSR is also used to specify the context size. The context size is fixed at 2072 bits (259 bytes). When loading context through the FIFO, all context data must be written prior to writing the context data size. The message data size must be written separately.

NOTE

In slave mode, when reloading an existing context, the user must write the context to the input FIFO, then write the context size (always 2072 bits). The write of the context size indicates to the that all context has been loaded. The user then writes the message data size to AFEUDSR. After this write, the user may begin writing message data to the FIFO.

Writing to AFEUDSR signals the AFEU to start processing data from the input FIFO as soon as it is available. If the value of data size is modified during processing, a context error will be generated.

NOTE

Writing to this register while in debug mode generates an illegal size error. Disable the illegal size error in the associated Interrupt Control Register prior to performing debug operations.

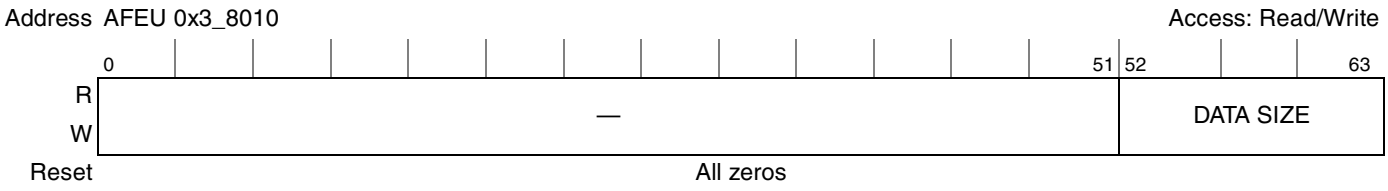


Figure 14-27. AFEU Data Size Register

14.5.3.5 AFEU Reset Control Register (AFEURCR)

The AFEU reset control register, shown in Figure 14-28, allows 3 levels reset that affect the AFEU only, as defined by 3 self-clearing bits. It should be noted that the AFEU executes an internal reset sequence for hardware reset, software reset, or module initialization, which performs proper initialization of the S-Box. To determine when this is complete, observe the RESET_DONE bit in AFEUSR.

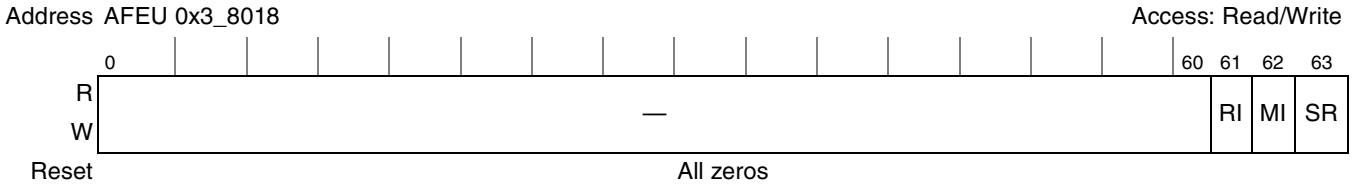


Figure 14-28. AFEU Reset Control Register (AFEURCR)

Table 14-22 describes AFEURCR fields

Table 14-22. AFEURCR Field Descriptions

Bits	Name	Description
0–60	—	Reserved
61	RI	Reset interrupt. Writing this bit causes AFEU interrupts signaling DONE and ERROR to be reset. It further resets the state of AFEUISR. 0 Do not reset 1 Reset interrupt logic
62	MI	Module initialization is nearly the same as software reset, except that the interrupt control register remains unchanged. 0 Do not reset 1 Reset most of AFEU
63	SR	Software reset is functionally equivalent to hardware reset (the RESET# pin), but only for AFEU. All registers and internal state are returned to their defined reset state. On negation of SW_RESET, the AFEU enters a routine to perform proper initialization of the S-box. 0 Do not reset 1 Full AFEU reset **Note:** Following a power on reset, the AFEU must be reset prior to first use. Failure to do so results in undefined behavior. Set the SR bit to perform AFEU reset.

14.5.3.6 AFEU Status Register (AFEUSR)

This status register, shown in Figure 14-29, contains 6 bits which reflect the state of the AFEU internal signals.

The AFEUSR is read-only. Writing to this location causes address error being reflected in AFEUISR.

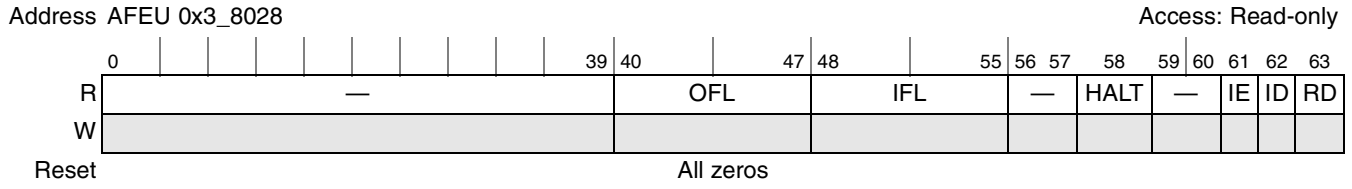


Figure 14-29. AFEU Status Register

Table 14-23 describes AFEUSR fields

Table 14-23. AFEUSR Field Descriptions

Bits	Name	Description
0–39	—	Reserved
40–47	OFL	The number of dwords currently in the output FIFO
48–55	IFL	The number of dwords currently in the input FIFO
56–57	—	Reserved
58	HALT	Halt. Indicates whether the AFEU has halted due to an error. 0 AFEU not halted 1 AFEU halted **Note:** Because the error causing the AFEU to stop operating may be masked to the interrupt status register, the status register is used to provide a second source of information regarding errors preventing normal operation.
59–60	—	Reserved
61	IE	Interrupt error. Reflects the state of the ERROR interrupt signal, as sampled by the controller interrupt status register (Section 14.7.2.2, “Interrupt Status Register (ISR)”). 0 AFEU is not signaling error 1 AFEU is signaling error
62	ID	Interrupt done. Reflects the state of the DONE interrupt signal, as sampled by the controller interrupt status register (Section 14.7.2.2, “Interrupt Status Register (ISR)”). 0 AFEU is not signaling done 1 AFEU is signaling done
63	RD	Reset done. 0 Reset in progress 1 Reset done. AFEU has completed its internal reset sequence.

14.5.3.7 AFEU Interrupt Status Register (AFEUISR)

The interrupt status register, seen in Figure 14-30, tracks the state of possible errors, if those errors are not masked, via the AFEU interrupt control register.

Address AFEU 0x3_8030 Access: Read-only

	0–50	51	52	53	54	55	56	57	58	59	60	61	62	63
R	—	IE	ERE	CE	KSE	DSE	ME	AE	OFE	IFE	—	IFO	OFU	—
W														
Reset	All zeros													

Figure 14-30. AFEU Interrupt Status Register (AFEUISR)

Table 14-24 describes AFEUISR fields.

Table 14-24. AFEUISR Field Descriptions

Bits	Names	Description
0–50	—	Reserved
51	IE	Internal error. An internal processing error was detected while performing encryption. 0 No error detected 1 Internal error
52	ERE	Early read error. The AFEU Context Memory or Control was read while the AFEU was performing encryption. 0 No error detected 1 Early read error
53	CE	Context error. AFEUMR, AFEUKR*n*, AFEUKSR, AFEUDSR, or context memory was modified while AFEU processes data. 0 No error detected 1 Context error
54	KSE	Key size error. A value outside the bounds 1–16 bytes was written to AFEUKSR. 0 No error detected 1 Key size error
55	DSE	Data size error. An inconsistent value (not a multiple of 8 bits, or larger than 64 bits) was written to AFEUDSR. 0 No error detected 1 Data size error
56	ME	Mode error. An illegal value was detected in AFEUMR. Note: Writing to reserved bits in mode register is likely source of error. 0 No error detected 1 Mode error
57	AE	Address error. An illegal read or write address was detected within the AFEU address space. 0 No error detected 1 Address error
58	OFE	Output FIFO error. The AFEU output FIFO was detected non-empty upon write of AFEUDSR. 0 No Output FIFO error detected 1 Output FIFO error detected
59	IFE	Input FIFO error. The AFEU input FIFO was detected non-empty upon generation of DONE interrupt. 0 No input FIFO error detected 1 Input FIFO error detected
60	—	Reserved
61	IFO	Input FIFO overflow. The AFEU input FIFO has been pushed while full. 0 No error detected 1 Input FIFO has overflowed. **Note:** When operating as a master, the AFEU implements flow-control, and FIFO size is not a limit to data input. When operated as a target, the AFEU cannot accept FIFO inputs larger than 512 bytes without overflowing.
62	OFU	Output FIFO underflow. The AFEU output FIFO has been read while empty. 0 No error detected 1 Output FIFO has underflow error.
63	—	Reserved

14.5.3.8 AFEU Interrupt Control Register (AFEUICR)

The interrupt control register, shown in Figure 14-31, controls the result of detected errors. For a given error (as defined in Section 14.5.3.7, "AFEU Interrupt Status Register (AFEUISR)"), if the corresponding AFEUICR bit is set, the error is disabled; no error interrupt occurs and AFEUISR is not updated to reflect the error. If the corresponding bit is not set, then upon detection of an error, AFEUISR is updated to reflect the error, causing assertion of the error interrupt signal, and causing the module to halt processing.

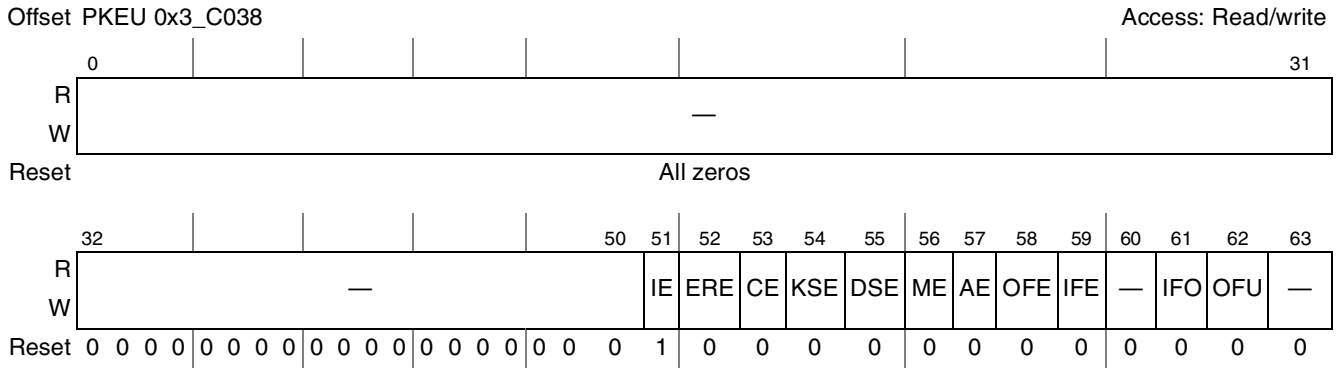


Figure 14-31. AFEU Interrupt Control Register (AFEUICR)

Table 14-25 describes AFEUICR fields.

Table 14-25. AFEUICR Field Descriptions

Bits	Names	Description
0–50	—	Reserved
51	IE	Internal error. An internal processing error was detected while performing encryption. 0 Internal error enabled 1 Internal error disabled
52	ERE	Early read error. The AFEU register was read while the AFEU was performing encryption. 0 Early read error enabled 1 Early read error disabled
53	CE	Context error. AFEUKR*n*, AFEUKSR, AFEUDSR, AFEUMR, or context memory was modified while AFEU was performing encryption. 0 Context error enabled 1 Context error disabled
54	KSE	Key size error. A value outside the bounds 1–16 bytes was written to AFEUKSR. 0 Key size error enabled 1 Key size error disabled
55	DSE	Data size error. An inconsistent value was written to the AFEUDSR. 0 Data size error enabled 1 Data size error disabled
56	ME	Mode error. An illegal value was detected in AFEUMR. 0 Mode error enabled 1 Mode error disabled
57	AE	Address error. An illegal read or write address was detected within the AFEU address space. 0 Address error enabled 1 Address error disabled

Table 14-25. AFEUICR Field Descriptions (continued)

Bits	Names	Description
58	OFE	Output FIFO error. The AFEU output FIFO was detected non-empty upon write of AFEUDSR. 0 Output FIFO non-empty error enabled 1 Output FIFO non-empty error disabled
59	IFE	Input FIFO error. The AFEU Input FIFO was detected non-empty upon generation of DONE interrupt. 0 Input FIFO non-empty error enabled 1 Input FIFO non-empty error disabled
60	—	Reserved
61	IFO	Input FIFO overflow. The AFEU Input FIFO has been pushed while full. 0 Input FIFO overflow error enabled 1 Input FIFO overflow error disabled
62	OFU	Output FIFO underflow. The AFEU Output FIFO has been read while empty. 0 Output FIFO underflow error enabled 1 Output FIFO underflow error disabled
63	—	Reserved

14.5.3.9 AFEU End of Message Register (AFEUEMR)

AFEUEMR, shown in Figure 14-32, is used to indicate an ARC4 operation may be completed. After the final message block is written to the input FIFO, AFEUEMR must be written. The value in AFEUDSR will be used to determine how many bits of the final message block (8–64, in multiples of 8) will be processed. Writing to this register causes the AFEU to process the final block of a message, allowing it to signal DONE. If AFEUMR[DC] is set (dump context mode is enabled), the context is written to the output FIFO following the last message word. A read of AFEUEMR always returns a zero value.

AFEUEMR is only used when the AFEU is operated as a slave. The descriptors and crypto-channel activate the AFEU (by means of an internally generated write to AFEUEMR) when the SEC acts as an initiator.

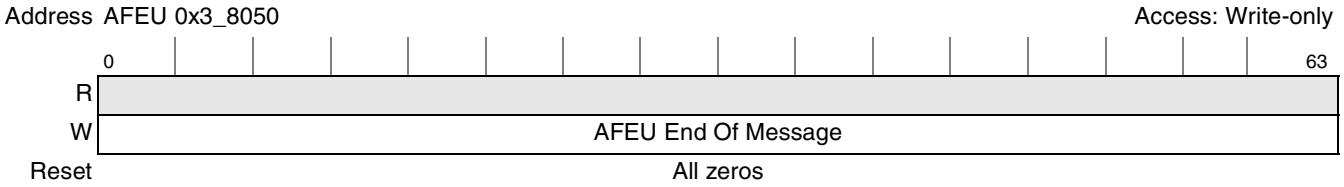


Figure 14-32. AFEU End of Message Register (AFEUEMR)

14.5.3.10 AFEU Context

This section provides additional information about the AFEU context memory and its related pointer register.

14.5.3.10.1 AFEU Context Memory

The S-Box memory consists of 32 64-bit words, each readable and writable. The S-Box contents should not be written with data unless it was previously read from the S-Box. Context data may only be written

if AFEUMR[PP] is set (prevent permutation mode is enabled, see Figure 14-25), and the context data must be written prior to the message data. If the context registers are written during message processing or AFEUMR[PP] is not set, a context error will be generated. Reading this memory while the module is not done will generate an error interrupt.

14.5.3.10.2 AFEU Context Memory Pointer Register

The context memory pointer register holds the internal context pointers that are updated with each byte of message processed. These pointers correspond to the values of I, J, and Sbox[I+1] in the ARC4 algorithm. If this register is written during message processing, a context error will be generated.

When performing ARC4 operations, the user has the option of performing a new S-Box permutation per packet, or unloading the contents of the S-box (context) and reloading this context prior to processing the next packet. The S-Box contents (256 bytes) plus the 3 bytes of the context memory pointers are unloaded and reloaded through the AFEU FIFOs.

AFEU context consists of the contents of the S-Box, as well as three counter values, which indicate the next values to be used from the S-Box. Context must be loaded in the same order in which it was unloaded.

14.5.3.11 AFEU Key Registers (AFEUK0, AFEUK1)

AFEU uses two write-only key registers to guide initial permutation of the AFEU S-Box, in conjunction with the AFEU key size register. AFEU performs permutation starting with the first byte of AFEUK0, and uses as many bytes from two key registers as necessary to complete the permutation. Reading either of these memory locations generates an address error interrupt.

14.5.3.11.1 AFEU FIFOs

AFEU uses an input FIFO/output FIFO pair to hold data before and after the encryption process. These FIFOs are multiply addressable, but those multiple addresses point only to the appropriate end of the appropriate FIFO. A write to anywhere in the AFEU FIFO address space causes the 64-bit-word to be pushed onto the AFEU input FIFO, and a read from anywhere in the AFEU FIFO address space causes a 64-bit-word to be popped off the AFEU output FIFO. Overflows and underflows caused by reading or writing the AFEU FIFOs are reflected in the AFEU interrupt status register.

14.5.4 Message Digest Execution Unit (MDEU)

The registers used in the MDEU are documented primarily for debug and slave mode operations. If the SEC requires the use of the MDEU when acting as an initiator, accessing these registers directly is unnecessary. The device drivers and the on-chip controller will abstract register level access from the user.

14.5.4.1 MDEU Mode Register (MDEUMR)

The MDEU mode register (MDEUMR) is used to program the function of the MDEU. Bits 56–63 of the MDEUMR are specified by the user through the MODE0 or MODE1 field of the descriptor header. The remaining bits are supplied by the channel and thus are not under direct user control.

The MDEUMR has two configurations, determined by the value of the NEW bit (see Figure 14-33 and Figure 14-34). The ‘old’ configuration (NEW = 0) is used by descriptor types numbered 0000_0, 0001_0, 0010_0, and 1100_0, and is backward compatible with previous versions of SEC 2.0. The ‘new’ configuration (NEW = 1) is used by all other descriptor types.

The MDEUMR is cleared when the MDEU is reset or re-initialized. Setting a reserved mode bit will generate a data error. If the mode register is modified during processing, a context error is generated.

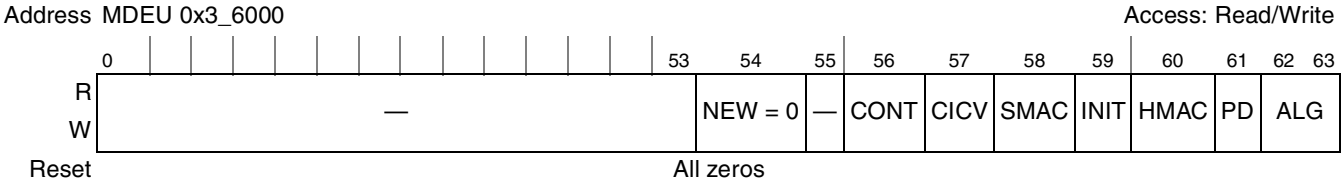


Figure 14-33. MDEU Mode Register (MDEUMR) in ‘Old’ Configuration

Table 14-26 describes MDEUMR fields in ‘old’ configuration.

Table 14-26. MDEUMR in ‘Old’ Configuration

Bits	Name	Description
The following bits are described for information only. They are not under direct user control.		
0–53	—	Reserved
54	NEW=0	Determines the configuration of the MDEU mode register (MDEUMR). This table shows the configuration for NEW = 0.
55	—	Reserved, must be set to zero
The following bits are controlled through the MODE0 or MODE1 fields of the descriptor header.		
56	CONT	Continue. Most operations will require this bit to be cleared. It is set only when the data to be hashed is spread across multiple descriptors. The value programmed in PD must be opposite to the value in this bit. 0 Do autopadding and complete the message digest. Used when the entire hash is performed with one descriptor, or on the last of a sequence of descriptors. 1 This hash will be continued in a subsequent descriptor. Do not autopad and do not complete the message digest.
57	CICV	Compare integrity check values 0 Normal operation; no ICV comparison 1 After the message digest (ICV) is computed, compare it to the data in the MDEU’s input FIFO. If the ICVs do not match, send an error interrupt to the channel. The number of bytes to be compared is given by the ICV size register. Only applicable to descriptor types that provide for reading an ICV in value.
58	SMAC	Specifies whether to perform an SSL-MAC operation 0 Normal operation 1 Perform an SSL3.0 MAC operation. This requires a key and key length. If this is set then the HMAC bit should be 0.

Table 14-26. MDEUMR in ‘Old’ Configuration (continued)

Bits	Name	Description
59	INIT	Initialization bit. Most operations will require this bit to be set. Cleared only for operations that load context from a known intermediate hash value. 0 Do not initialize digest registers. In this case the registers must be loaded from a hash context pointer in the descriptor. When the data to be hashed is spread across multiple descriptors, this bit must be 0 on all but the first descriptor. 1 Do an algorithm-specific initialization of the digest registers.
60	HMAC	Specifies whether to perform an HMAC operation 0 Normal operation 1 Perform an HMAC operation. This requires a key and key length. If this is set then the SMAC bit should be 0.
61	PD	This bit must be programmed opposite to the CONT bit.
62–63	ALG	Message digest algorithm selection 00 SHA-160 algorithm (full name for SHA-1) 01 SHA-256 algorithm 10 MD5 algorithm 11 SHA-224 algorithm

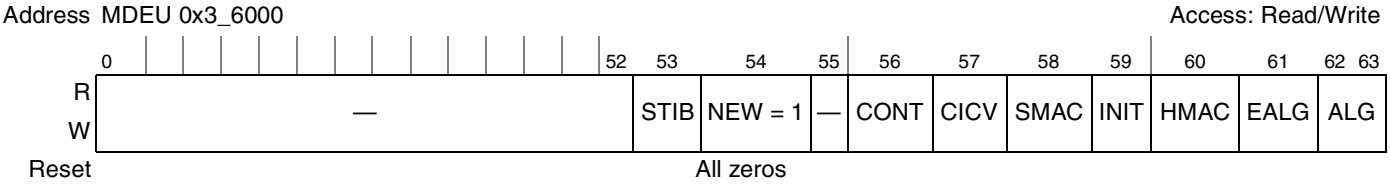


Figure 14-34. MDEU Mode Register (MDEUMR) in ‘New’ Configuration

Table 14-27 describes MDEUMR fields in ‘new’ configuration.

Table 14-27. MDEUMR in ‘New’ Configuration

Bits	Name	Description
The following bits are described for information only. They are not under direct user control.		
0–52	—	Reserved
53	STIB	SSL/TLS inbound, block cipher 0 Normal operation. 1 Special operation only for SSL/TLS inbound, block cipher. Upon receiving end-of-message, the MDEU performs a calculation involving the last valid byte of data written into its input FIFO (which is Pad Length) to compute a final data size. The MDEU then processes the amount of data specified by this data size, and completes the message digest.
54	NEW=1	Determines the configuration of the MDEU mode register (MDEUMR). This table shows the configuration for NEW=1.
55	—	Reserved, must be set to zero
The following bits are controlled through the MODE0 or MODE1 fields of the descriptor header.		

Table 14-27. MDEUMR in 'New' Configuration (continued)

Bits	Name	Description
56	CONT	Continue. Most operations will require this bit to be cleared. Set only when the data to be hashed is spread across multiple descriptors. 0 Do autopadding and complete the message digest. Used when the entire hash is performed with one descriptor, or on the last of a sequence of descriptors. 1 This hash will be continued in a subsequent descriptor. Do not autopad and do not complete the message digest.
57	CICV	Compare integrity check values 0 Normal operation; no ICV comparison. 1 After the message digest (ICV) is computed, compare it to the data in the MDEU's input FIFO. If the ICVs do not match, send an error interrupt to the channel. The number of bytes to be compared is given by the ICV size register.
58	SMAC	Specifies whether to perform an SSL-MAC operation 0 Normal operation 1 Perform an SSL3.0 MAC operation. This requires a key and key length. If this is set then the HMAC bit should be 0.
59	INIT	Initialization bit. Most operations will require this bit to be set. Cleared only for operations that load context from a known intermediate hash value. 0 Do not initialize digest registers. In this case the registers must be loaded from a hash context pointer in the descriptor. When the data to be hashed is spread across multiple descriptors, this bit is set on all but the first descriptor. 1 Do an algorithm-specific initialization of the digest registers.
60	HMAC	Specifies whether to perform an HMAC operation 0 Normal operation 1 Perform an HMAC operation. This requires a key and key length. If this is set then the SMAC bit should be 0.
61	EALG	The EALG (Extended Algorithm bit) and ALG (Algorithm) bits together specify the message digest algorithm, as follows: 000 SHA-160 algorithm (full name for SHA-1) 001 SHA-256 algorithm 010 MD5 algorithm 011 SHA-224 algorithm Others: Reserved
62–63	ALG	

14.5.4.2 Recommended Settings for MDEUMR

The most common task likely to be executed through the MDEU is HMAC generation. HMACs are used to provide message integrity within a number of security protocols, including IPSec, and TLS. The SSL 3.0 protocol uses a slightly different 'SSL-MAC'. If an HMAC or SSL-MAC is to be performed using a single descriptor (with the MDEU acting as sole or secondary EU), the following mode register bit settings should be used:

Table 14-28. Mode Register—HMAC or SSL-MAC Generated by Single Descriptor

Bits	Field	Value	
		for HMAC	for SSL-MAC
56	CONT	0 (off)	0 (off)
58	SMAC	0(on)	1(on)
59	INIT	1(on)	1(on)
60	HMAC	1(on)	0(on)

To generate an HMAC for a message that is spread across a sequence of descriptors, the following mode register bit settings should be used:

Table 14-29. Mode Register—HMAC Generated across a Sequence of Descriptors

Bits	Field	Value		
		First Descriptor	Middle Descriptor(s)	Final Descriptor
56	CONT	1 (on)	1 (on)	0 (off)
59	INIT	1 (on)	0 (off)	0 (off)
60	HMAC	1 (on)	0 (off)	1 (on)

All descriptors other than the final descriptor must output the intermediate message digest for the next descriptor to reload as MDEU context.

SSL-MAC operations cannot be spread across a sequence of descriptors.

Additional information on descriptors can be found in Section 14.2.1, “Data Packet Descriptors.”

14.5.4.3 MDEU Key Size Register (MDEUKSR)

MDEUKSR, shown in Figure 14-35, indicates the number of bytes of key memory that should be used in HMAC generation. The MDEU supports at most 64 bytes of key. The MDEU will generate a key size error if the value written to MDEUKSR exceeds 64 bytes.

NOTE

Writing to this register while in debug mode generates an illegal size error. Disable the illegal size error in the associated interrupt control register before performing debug operations.

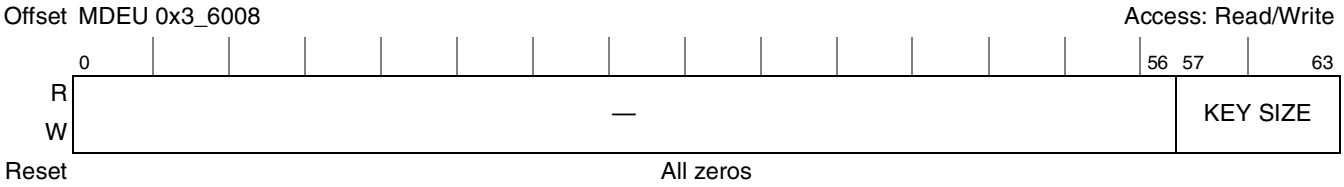


Figure 14-35. MDEU Key Size Register (MDEUKSR)

14.5.4.4 MDEU Data Size Register (MDEUDSR)

The MDEU data size register, shown in Figure 14-36, stores the size of the last block of data (in bits) to be processed. Since the MDEU does not support bit offsets, any value other than 0 in bits 61–63 will cause a data size error. Bits 58–60 are used to identify the ending byte location in the last 8-byte dword. This is used to add the data padding when auto padding is selected. MDEUDSR is cleared when the MDEU is reset, re-initialized, and at the end of processing the complete message.

NOTE

Writing to MDEUDSR allows the MDEU to enter auto-start mode. Therefore, the required context data should be written prior to writing the data size.

Writing to this register while in debug mode generates an illegal size error. Disable the illegal size error in the associated interrupt control register before performing debug operations.

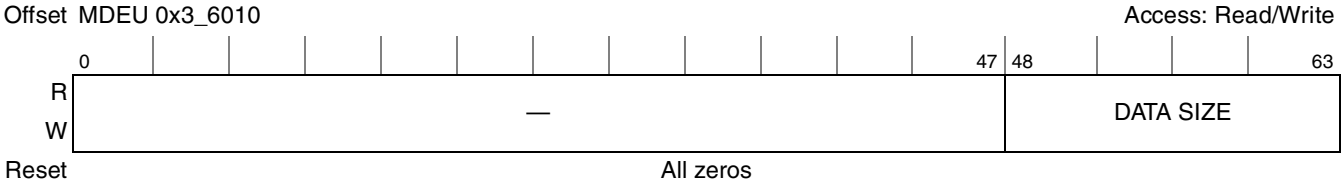


Figure 14-36. MDEU Data Size Register (MDEUDSR)

14.5.4.5 MDEU Reset Control Register (MDEURCR)

MDEURCR, shown in Figure 14-37, allows three levels reset of just the MDEU, as defined by the three self-clearing bits.

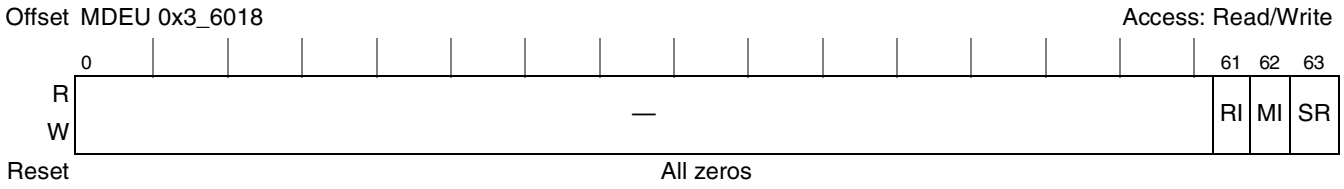


Figure 14-37. MDEU Reset Control Register (MDEURCR)

Table 14-30 describes MDEU reset control register fields.

Table 14-30. MDEU Reset Control Register Field Descriptions

Bits	Name	Description
0–60	—	Reserved
61	RI	Reset interrupt. Writing this bit active high causes MDEU interrupts signaling DONE and ERROR to be reset. It further resets the state of the MDEUISR. 0 No reset 1 Reset interrupt logic

Table 14-30. MDEU Reset Control Register Field Descriptions (continued)

Bits	Name	Description
62	MI	Module initialization is nearly the same as software reset, except that MDEUICR is unchanged. 0 No reset 1 Reset most of MDEU
63	SR	Software reset is functionally equivalent to hardware reset (the RESET# pin), but only for the MDEU. All registers and internal state are returned to their defined reset state. 0 No reset 1 Full MDEU reset

14.5.4.6 MDEU Status Register (MDEUSR)

MDEUSR reflects the state of the MDEU internal signals. The majority of these internal signals reflect the state of low-level MDEU functions, such as data padding and key padding, and are not important to the user; however, the user should be aware that reads of this register, especially during processing, are likely to return non-zero values for many bits between 0–57. The four signals shown are those most likely to be of interest to the user.

MDEUSR, shown in Figure 14-38, is read only.

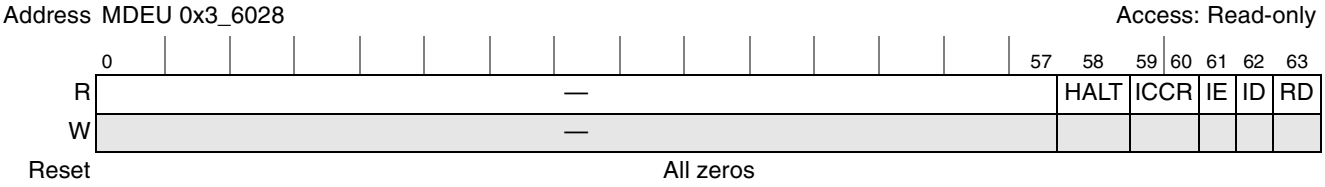


Figure 14-38. MDEU Status Register

Table 14-23 describes MDEUSR fields.

Table 14-31. MDEU Status Register Field Descriptions

Bits	Name	Description
0–57	—	Reserved
58	HALT	Halt. Indicates that the MDEU halted due to an error. 0 MDEU not halted 1 MDEU halted **Note:** Because the error causing the MDEU to stop operating may be masked to MDEUISR, MDEUSR is used to provide a second source of information regarding errors preventing normal operation.
59–60	ICCR	Integrity check comparison result 00 No integrity check comparison was performed. 01 The integrity check comparison passed. 10 The integrity check comparison failed. 11 Reserved **Note:** A passed or failed result is generated only if ICV checking is enabled.

Table 14-31. MDEU Status Register Field Descriptions (continued)

Bits	Name	Description
61	IE	Interrupt error. This status bit reflects the state of the ERROR interrupt signal, as sampled by the controller ISR (Section 14.7.2.2, “Interrupt Status Register (ISR)”). 0 MDEU is not signaling error. 1 MDEU is signaling error.
62	ID	Interrupt done. This status bit reflects the state of the DONE interrupt signal, as sampled by the controller ISR (Section 14.7.2.2, “Interrupt Status Register (ISR)”). 0 MDEU is not signaling done 1 MDEU is signaling done
63	RD	Reset done. This status bit, when high, indicates that MDEU has completed its internal reset sequence. 0 Reset in progress 1 Reset done

14.5.4.7 MDEU Interrupt Status Register (MDEUISR)

The interrupt status register tracks the state of possible errors, if those errors are not masked through the MDEUISR. The definition of each field in MDEUISR is shown in Figure 14-39.

Address MDEU 0x3_6030 Access: Read-only

	0–48	49	50	51	52	53	54	55	56	57	58–60	61	62–63
R	—	ICE	—	IE	ERE	CE	KSE	DSE	ME	AE	—	IFO	—
W													
Reset	All zeros												

Figure 14-39. MDEU Interrupt Status Register (MDEUISR)

Table 14-32 describes MDEUISR fields.

Table 14-32. MDEUISR Field Descriptions

Bits	Name	Description
0–48	—	Reserved
49	ICE	Integrity check error. 0 No error detected 1 Integrity check error detected. An ICV check was performed and the supplied ICV did not match the one computed by the MDEU.
50	—	Reserved
51	IE	Internal error. Indicates the MDEU has been locked up and requires a reset before use. 0 No internal error detected 1 Internal error detected **Note:** This bit is set any time an enabled error condition occurs and can only be cleared by resetting the MDEU.
52	ERE	Early read error. The MDEU context was read before the MDEU completed the hashing operation. 0 No error detected 1 Early read error

Table 14-32. MDEUISR Field Descriptions (continued)

Bits	Name	Description
53	CE	Context error. The MDEU key register, MDEUKSR, or MDEUDSR was modified while MDEU was hashing. 0 No error detected 1 Context error
54	KSE	Key size error. A value greater than 64 bytes was written to MDEUKSR. 0 No error detected 1 Key size error
55	DSE	Data size error. A value not a multiple of 512 bits while the MDEU mode register autopad bit is cleared. 0 No error detected 1 Data size error
56	ME	Mode error. An illegal value for ALG was detected in MDEUMR. 0 No error detected 1 Mode error
57	AE	Address error. An illegal read or write address was detected within the MDEU address space. 0 No error detected 1 Address error
58–60	—	Reserved
61	IFO	Input FIFO overflow. The MDEU input FIFO has been pushed while full. 0 No overflow detected 1 Input FIFO has overflowed **Note:** When operating as a master, the implements flow-control, and FIFO size is not a limit to data input. When operated as a target, the cannot accept FIFO inputs larger than 512 Bytes without overflowing.
62–63	—	Reserved

14.5.4.8 MDEU Interrupt Control Register (MDEUICR)

MDEUICR, shown in Figure 14-40, controls the result of detected errors. For a given error (as defined in Section 14.5.4.7, “MDEU Interrupt Status Register (MDEUISR)”), if the corresponding bit in this register is set, then the error is disabled; no error interrupt occurs and the interrupt status register is not updated to reflect the error. If the corresponding bit is not set, then upon detection of an error, the interrupt status register is updated to reflect the error, causing assertion of the error interrupt signal, and causing the module to halt processing.

Address MDEU 0x3_6038 Access: Read/Write

	0–48	49	50	51	52	53	54	55	56	57	58–60	61	62	63
R W	—	ICE	—	IE	ERE	CE	KSE	DSE	ME	AE	—	IFO	—	
Reset	All zeros	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

Figure 14-40. MDEU Interrupt Control Register (MDEUICR)

Table 14-32 describes MDEUICR fields.

Table 14-33. MDEUICR Field Descriptions

Bits	Name	Description
0–48	—	Reserved
49	ICE	Integrity check error. The supplied ICV did not match the one computed by the MDEU. 0 Integrity check error enabled 1 Integrity check error disabled
50	—	Reserved
51	IE	Internal error. An internal processing error is detected while performing hashing. 0 Internal error enabled 1 Internal error disabled
52	ERE	Early read error. The MDEU register is read while the MDEU is performing hashing. 0 Early read error enabled 1 Early read error disabled
53	CE	Context error. The MDEU key register, MDEUKSR, MDEUDSR, or MDEUMR is modified while the MDEU is performing hashing. 0 Context error enabled 1 Context error disabled
54	KSE	Key size error. A value outside the bounds of 64 bytes is written to the MDEU key size register 0 Key size error enabled 1 Key size error disabled
55	DSE	Data size error. An inconsistent value is written to MDEUDSR: 0 Data size error enabled 1 Data size error disabled
56	ME	Mode error. An illegal value is detected in MDEUMR. 0 Mode error enabled 1 Mode error disabled
57	AE	Address error. An illegal read or write address is detected within the MDEU address space. 0 Address error enabled 1 Address error disabled
58–60	—	Reserved
61	IFO	Input FIFO overflow. The MDEU input FIFO has been pushed while full. 0 Input FIFO overflow error enabled 1 Input FIFO overflow error disabled
62–63	—	Reserved

14.5.4.9 MDEU ICV Size Register (MDEUICVSR)

The MDEU ICV size register, shown in Figure 14-41, stores the number of bytes of the ICV result to be compared if the MDEU performs ICV comparison. (See Section 14.5.4.1, “MDEU Mode Register (MDEUMR).”)

This register is cleared when the MDEU is reset or re-initialized.

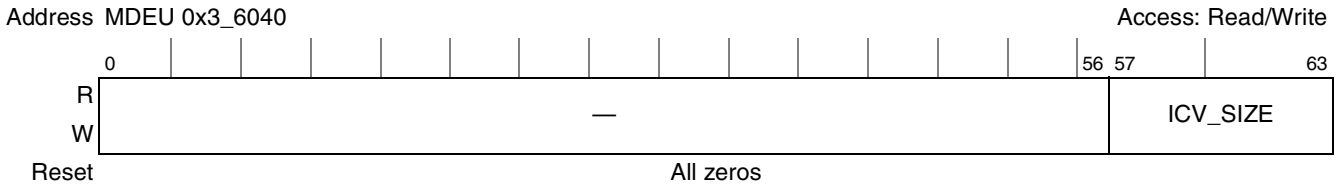


Figure 14-41. MDEU ICV Size Register

14.5.4.10 MDEU EU-Go Register (MDEUEUG)

The EU-Go register in the MDEU, see Figure 14-42, is used to indicate an authentication operation may be completed. After the final message block is written to the input FIFO, the EU-Go register must be written. The value in the data size register will be used to determine how many bits of the final message block (always 512) will be processed. Note that this register has no data size, and during the write operation, the host data bus is not read. Hence, any data value is accepted. Normally, a write operation with a zero data value is performed. Moreover, no read operation from this register is meaningful, but no error is generated, and a zero value is always returned. Writing to this register is merely a trigger causing the MDEU to process the final block of a message, allowing it to signal DONE.

The DEU EU-Go register is only used when the SEC is operated as a slave. The descriptors and crypto-channel activate the MDEU (via an internally generated write to the EU-Go register) when the SEC acts as an initiator.

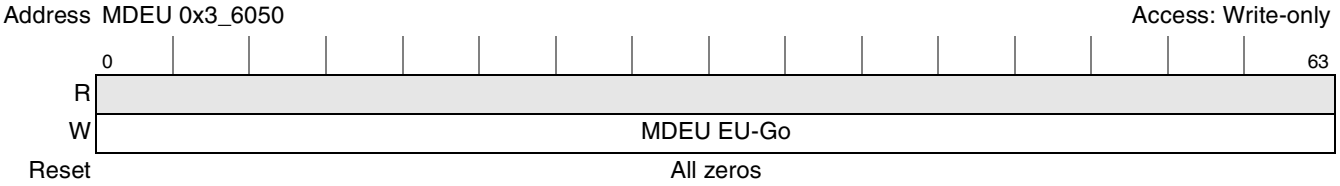


Figure 14-42. MDEU EU-Go Register

14.5.4.11 MDEU Context Registers

For MDEU, context consists of the hash plus the message length count. Write access to this register block allows continuation of a previous hash. Reading these registers provide the resulting message digest or HMAC, along with an aggregate bit count.

NOTE

SHA-1and SHA-256 are big endian. MD5 is little endian. The MDEU module internally reverses the endianness of the five registers A, B, C, D, and E upon writing to or reading from the MDEU context if the MDEU mode register indicates MD5 is the hash of choice. Most other endian considerations are performed as 8-byte swaps. In this case, 4-byte endianness swapping is performed within the A, B, C, D, and E fields as individual registers. Reading this memory location while the module is not done will generate an error interrupt.

After a power-on reset, all the MDEU context register values are cleared. Figure 14-43 shows how the MDEU context registers are initialized if the INIT bit is set in the MDEU mode register. All registers are

initialized, regardless of mode selected, however only the appropriate context register values are used in hash generation per the mode selected. The user typically doesn't care about the MDEU Context Register initialization values, however they are documented for completeness in the event the user reads these registers during a debug operation. MDEU reset via the MDEU reset control register (Figure 14-37) or SEC global software reset (Figure 14-71) does not clear these registers.

	0 – 31	32 – 63	
Name	A	B	Context offset 0x3_6100
MD-5	0x01234567	0x89ABCDEF	
SHA-1	0x67452301	0xEFCDAB89	
SHA-256	0x6A09E667	0xBB67AE85	
Name	C	D	Context offset 0x3_6108
MD-5	0xFEDCBA98	0x76543210	
SHA-1	0x98BADCFE	0x10325476	
SHA-256	0x3C6EF372	0xA54FF53A	
Name	E	F	Context offset 0x3_6110
MD-5	0xF0E1D2C3	0x8C68059B	
SHA-1	0xC3D2E1F0	0x9B05688C	
SHA-256	0x510E527F	0x9B05688C	
Name	G	H	Context offset 0x3_6118
MD-5	0xABD9831F	0x19CDE05B	
SHA-1	0x1F83D9AB	0x5BE0CD19	
SHA-256	0x1F83D9AB	0x5BE0CD19	
Name	Message Length Count		Context offset 0x3_6120
Reset	0		

Figure 14-43. MDEU Context Register

14.5.4.12 MDEU Key Registers

The MDEU maintains eight 64-bit registers for writing an HMAC key. The IPAD and OPAD operations are performed automatically on the key data when required.

NOTE

SHA-1 and SHA-256 are big endian. MD5 is little endian. The MDEU module internally reverses the endianness of the key upon writing to or reading from the MDEU key registers if the MDEU mode register indicates MD5 is the hash of choice.

14.5.4.13 MDEU FIFO

MDEU uses an input FIFO to hold data to be hashed. The input FIFO is multiply addressable, but those multiple addresses point only to the write (push) end of the FIFO. A write to anywhere in the MDEU FIFO address space causes the 64-bit-words to be pushed onto the MDEU input FIFO, and a read from anywhere in the MDEU FIFO address space returns all zeros.

NOTE

SHA-1 and SHA-256 are big endian. MD5 is little endian. The MDEU module internally reverses the endianness of the key upon writing to or reading from the MDEU key registers if the MDEU mode register indicates MD5 is the hash of choice.

14.5.5 Random Number Generator (RNG)

This section contains details about the random number generator (RNG), including detailed register map, modes of operation, status and control registers, and FIFOs.

The RNG is an execution unit capable of generating 64-bit random numbers. It is designed to comply with the FIPS-140 standard for randomness and non-determinism. A linear feedback shift register (LSFR) and cellular automata shift register (CASR) are operated in parallel to generate pseudo-random data.

The RNG consists of six major functional blocks:

- Bus interface unit (BIU)
- Linear feedback shift register (LFSR)
- Cellular automata shift register (CASR)
- Clock controller
- Six ring oscillators

The states of the LFSR and CASR are advanced at unknown frequencies determined by the two ring oscillator clocks and the clock control. When a read is performed, the oscillator clocks are halted and a collection of bits from the LFSR and CASR are XORed together to obtain the 64-bit random output.

The registers used in the MDEU are documented primarily for debug and slave mode operations. If the SEC requires the use of the MDEU when acting as an initiator, accessing these registers directly is unnecessary. The device drivers and the on-chip controller will abstract register level access from the user.

14.5.5.1 RNG Mode Register (RNGMR)

RNGMR, shown in Figure 14-44, is used to control the RNG. One operational mode, randomizing, is defined. Writing any other value than 0 to 56:63 results in a data error interrupt reflected in the RNG interrupt status register. The mode register also reflects the value of burst size, which is loaded by the crypto-channel during normal operation with the as an initiator. Burst size is not relevant to slave mode operations, where an external host pushes and pulls data from the execution units.

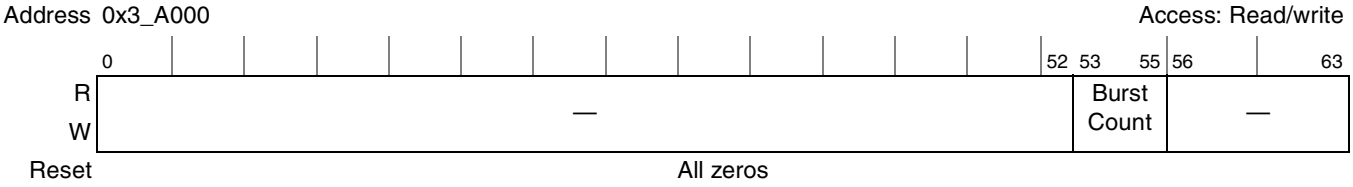


Figure 14-44. RNG Mode Register

Table 14-34. RNG Mode Register Definitions

Bits	Name	Description
0–52	—	Reserved, must be set to zero.
53–55	Burst Count	Burst Count. Implements flow control to allow larger than FIFO sized blocks of data to be processed with a single key/context. The RNG signals to the crypto-channel that a 'Burst Size' amount of data is available to be pulled from the FIFO. **Note:** The inclusion of this field in the RNG mode register is to avoid confusing a user who may read this register in debug mode. Burst Size should not be written directly to the RNG.
56–63	—	Reserved

14.5.5.2 RNG Data Size Register (RNGDSR)

RNGDSR, shown in Figure 14-45, is used to tell the RNG to begin generating random data. The actual contents of the data size register does not affect the operation of the RNG. After a reset and prior to the first write of data size, the RNG builds entropy without pushing data onto the FIFO. Once the data size register is written, the RNG will begin pushing data onto the FIFO. Data will be pushed onto the FIFO every 256 cycles until the FIFO is full. The RNG then attempts to keep the FIFO full.

NOTE

Writing to this register while in debug mode generates an illegal size error. Disable the illegal size error in the associated Interrupt Control Register prior to performing debug operations.

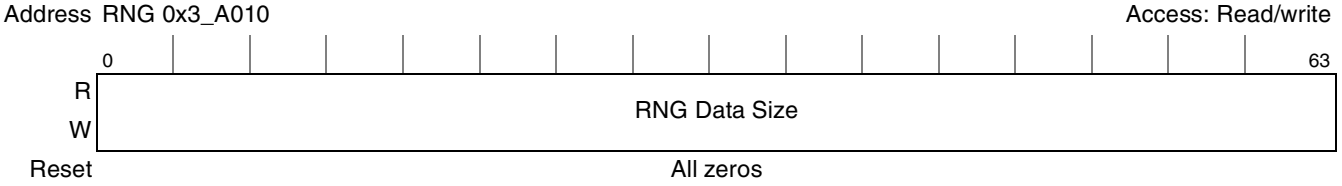


Figure 14-45. RNG Data Size Register

14.5.5.3 RNG Reset Control Register (RNGRCR)

RNGRCR, shown in Figure 14-46, contains three reset options specific to the RNG.

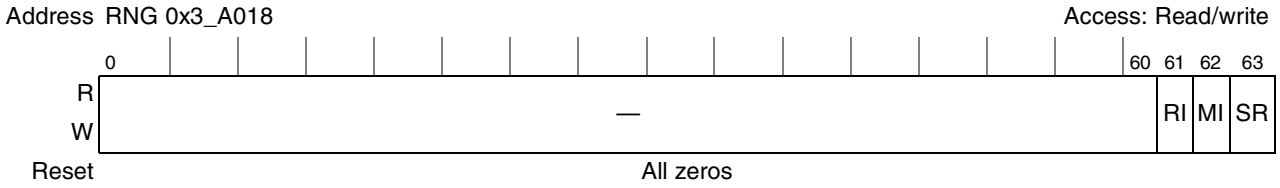


Figure 14-46. RNG Reset Control Register

Table 14-35 describes RNG reset control register fields.

Table 14-35. RNG Reset Control Register Field Descriptions

Bits	Name	Description
0–60	—	Reserved
61	RI	Reset Interrupt. Writing this bit active high causes RNG interrupts signalling DONE and ERROR to be reset. It further resets the state of the RNG interrupt status register. 0 No reset 1 Reset interrupt logic
62	MI	Module Initialization. This reset value performs enough of a reset to prepare the RNG for another request, without forcing the internal control machines and the output FIFO to be reset, thereby invalidating stored random numbers or requiring reinvocation of a warm-up period. Module initialization is nearly the same as software reset, except that the interrupt control register remains unchanged. 0 No reset 1 Reset most of RNG
63	SR	Software reset is functionally equivalent to hardware reset (the RESET# pin), but only for the RNG. All registers and internal state are returned to their defined reset state. 0 No reset 1 Full RNG reset

14.5.5.4 RNG Status Register (RNGSR)

RNGSR, shown in Figure 14-47, contains 6 fields which reflect the state of the RNG internal signals.

Writing to this location causes an address error being reflected in the RNG interrupt status register.

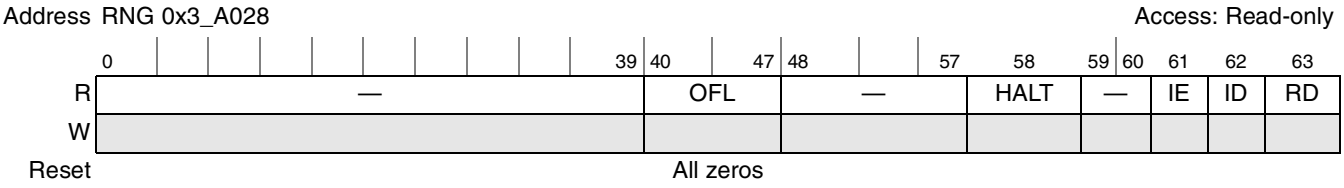


Figure 14-47. RNG Status Register

Table 14-23 describes RNG status register fields.

Table 14-36. RNG Status Register Field Descriptions

Bits	Name	Description
0–39	—	Reserved
40–47	OFL	The number of dwords currently in the output FIFO
48–57	—	Reserved. Internal status bits may be observed as non-zero.
58	HALT	Halt. Indicates that the RNG has halted due to an error. 0 RNG not halted 1 RNG halted Note: Because the error causing the RNG to stop operating may be masked to the interrupt status register, the status register is used to provide a second source of information regarding errors preventing normal operation.
59–60	—	Reserved
61	IE	Interrupt Error. This status bit reflects the state of the ERROR interrupt signal, as sampled by the Controller Interrupt Status Register (Section 14.7.2.2, “Interrupt Status Register (ISR)”). 0 RNG is not signaling error 1 RNG is signaling error
62	ID	Interrupt done. This status bit reflects the state of the DONE interrupt signal, as sampled by the Controller Interrupt Status Register (Section 14.7.2.2, “Interrupt Status Register (ISR)”). 0 RNG is not signaling done 1 RNG is signaling done
63	RD	Reset Done. This status bit, when high, indicates that the RNG has completed its internal reset sequence. 0 Reset in progress 1 Reset done

14.5.5.5 RNG Interrupt Status Register (RNGISR)

RNGISR, shown in Figure 14-48, tracks the state of possible unmasked errors via RNGICR.

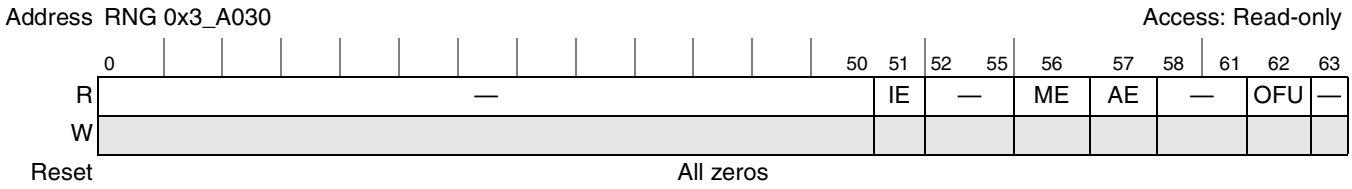


Figure 14-48. RNG Interrupt Status Register

Table 14-37 describes RNGISR fields.

Table 14-37. RNG Interrupt Status Register Field Descriptions

Bits	Name	Description
0–50	—	Reserved
51	IE	Internal Error 0 No internal error detected 1 Internal error

Table 14-37. RNG Interrupt Status Register Field Descriptions (continued)

Bits	Name	Description
52–55	—	Reserved
56	ME	Mode Error. Indicates that the host has attempted to write an illegal value to the mode register 0 = Valid data 1 = Invalid data error
57	AE	Address Error. An illegal read or write address was detected within the RNG address space. 0 No error detected 1 Address error
58–61	—	Reserved
62	OFU	Output FIFO Underflow. The RNG Output FIFO has been read while empty. 0 No overflow detected 1 Output FIFO has underflowed
63	—	Reserved

14.5.5.6 RNG Interrupt Control Register (RNGICR)

RNGICR controls the result of detected errors. For a given error (as defined in Section 14.5.5.5, “RNG Interrupt Status Register (RNGISR)”), if the corresponding RNGICR bit is set, the error is disabled; no error interrupt occurs and the interrupt status register is not updated to reflect the error. If the corresponding bit is not set, then upon detection of an error, RNGISR is updated to reflect the error, causing assertion of the error interrupt signal, and causing the module to halt processing.

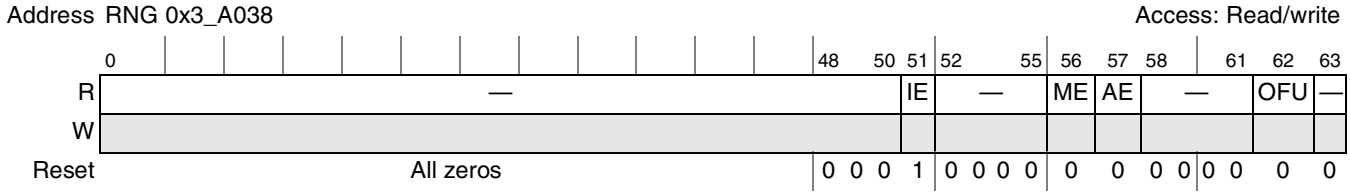


Figure 14-49. RNG Interrupt Control Register

Table 14-38 describes RNG interrupt status register fields.

Table 14-38. RNG Interrupt Control Register Field Descriptions

Bits	Name	Description
0–50	—	Reserved
51	IE	Internal Error. An internal processing error was detected while generating random numbers. 0 Internal error enabled 1 Internal error disabled
52–55	—	Reserved
56	ME	Mode Error. An illegal value was detected in the mode register. 0 Mode error enabled 1 Mode error disabled

Table 14-38. RNG Interrupt Control Register Field Descriptions (continued)

Bits	Name	Description
57	AE	Address Error. An illegal read or write address was detected within the MDEU address space. 0 Address error enabled 1 Address error disabled
58–61	—	Reserved
62	OFU	Output FIFO Underflow. RNG Output FIFO has been read while empty. 0 Output FIFO underflow error enabled 1 Output FIFO underflow error disabled
63	—	Reserved

14.5.5.7 RNG EU-Go Register (RNGEUG)

RNGEUG, shown in Figure 14-50, is a writable location but serves no function in the RNG. It is documented for the sake of consistency with the other EUs.

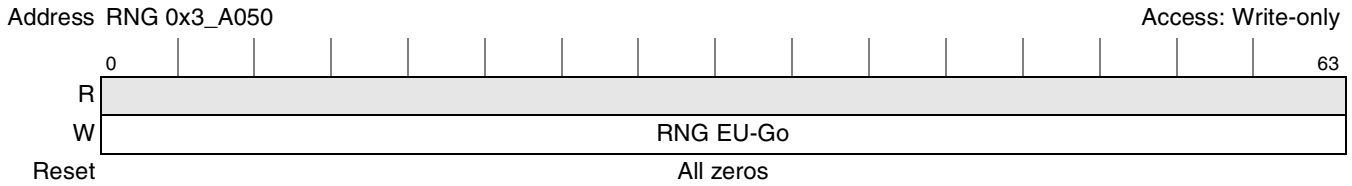


Figure 14-50. RNG EU-Go Register

14.5.5.8 RNG FIFO

RNG uses an output FIFO to collect periodically sampled random 64-bit-words, with the intent that random data always be available for reading. The FIFO is multiply addressed, but those multiple addresses point only to the appropriate end of the output FIFO. A read from anywhere in the RNG FIFO address space causes a 64-bit-word to be popped off of the RNG output FIFO. Underflows caused by reading or writing the RNG output FIFO are reflected in the RNG interrupt status register. Also, a write to the RNG output FIFO space will be reflected as an addressing error in the RNG interrupt status register.

14.5.6 Advanced Encryption Standard Execution Units (AESU)

This section contains details about the Advanced Encryption Standard Execution Units (AESU), including detailed register map, modes of operation, status and control registers, and FIFOs. The registers used in the AESU are documented primarily for debug and slave mode operations. If the SEC requires the use of the AESU when acting as an initiator, accessing these registers directly is unnecessary. The device drivers and the on-chip controller will abstract register level access from the user.

14.5.6.1 AESU Mode Register (AESUMR)

The AESU mode register, shown in Figure 14-51, contains 7 bits which are used to program the AESU. The mode register is cleared when the AESU is reset or re-initialized. Setting a reserved mode bit will generate a data error. If the mode register is modified during processing, a context error will be generated.

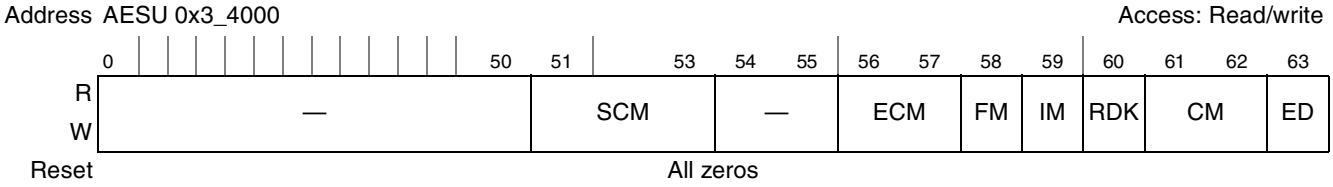


Figure 14-51. AESU Mode Register

Table 14-39 describes AESU mode register fields.

Table 14-39. AESU Mode Register Field Descriptions

Bits	Name	Description
The following bits are described for information only. They are not under direct user control.		
0–50	—	Reserved
51–53	SCM	Sub-cipher-mode. Specifies additional options specific to particular cipher modes. • XOR cipher mode: specifies the number of sources to be XORed together. Valid values are 2 and 3. • For all other cipher modes, this field must be 0.
54–55	—	Reserved, must be set to zero.
The following bits are controlled through the MODE0 field of the descriptor header.		
56–57	ECM	Extend cipher mode. Used in combination with bits 61:62 “Cipher Mode” to define the mode of AES operation. See Table 14-40 for mode bit combinations.
58	FM	Final MAC (FM). Processes final message block and generates final MAC tag at end of message processing (CCM mode only) 0 Do not generate final MAC tag 1 Generate final MAC tag after CCM processing is complete.
59	IM	Initialize MAC(IM). Initializes AESU for new message (CCM mode only) 0 Do not initialize (context will be loaded by host) 1 Initialize new message with nonce
60	RDK	Restore decrypt key (RDK). Specifies that key data write will contain pre-expanded key (decrypt mode only). See note below on use of RDK bit. 0 Expand the user key prior to decrypting the first block 1 Do not expand the key. The expanded decryption key will be written following the context switch.
61–62	CM	Cipher mode. Used in combination with bits 56:57 “Extend Cipher Mode” to define the mode of AES operation. See Table 14-40 for mode bit combinations.
63	ED	Encrypt/Decrypt. If set, AESU operates the encryption algorithm; if not set, AESU operates the decryption algorithm. 0 Perform decryption 1 Perform encryption Note: This bit is ignored if CM is set to “11” - CTR Mode.

SRT is not a new AES mode, it is an AESU method of performing AES-CTR mode with reduced context loading overhead specifically for performing SRTP. It should be used with descriptor type 0010_0’srtp’. See Section Figure 14-59., “AESU Context Register,” for more information on how SRT mode reduces context loading overhead.

Table 14-40. AES Cipher Modes

Mode	ECM (56:57)	CM (61:62)
ECB	00	00
CBC	00	01
Res	xx	10
CTR	00	11
SRT[1]	01	11
CCM (without ICV comparison)	10	00
CCM (with ICV comparison)	11	00
XOR	11	11
All Others	Reserved	

[1] SRT is not a new AES mode, it is an AESU method of performing AES-CTR mode with reduced context loading overhead specifically for performing SRTP. It should be used with descriptor type 0010_0'srtp'. See Section 14.5.6.9.3, "Context for SRT Mode," for more information on how SRT mode reduces context loading overhead.

NOTE

Restore decrypt key (RDK)—In most networking applications, the decryption of an AES protected packet will be performed as a single operation. However, if circumstances dictate that the decryption of a message should be split across multiple descriptors, the AESU allows the user to save the decrypt key, and the active AES context, to memory for later re-use. This saves the internal AESU processing overhead associated with regenerating the decryption key schedule (~12 AESU clock cycles for the first block of data to be decrypted.)

The use of RDK is completely optional, as the Input time of the preserved decrypt key may exceed the ~12 cycles required to restore the decrypt key for processing the first block.

To use RDK, the following procedure is recommended:

The descriptor type used in decryption of the first portion of the message is "0100_0- AESU Key Expand Output". The AESU mode must be "Decrypt". See Table 14-6. for more information. The descriptor will cause the SEC to write the contents of the Context registers and the key registers (containing the expanded decrypt key) to memory.

To process the remainder of the message, use a 'common' descriptor type (0001_0), and set the restore-decrypt-key mode bit. Load the context registers and the expanded decrypt key with previously saved key and context data from the first message. The key size is written as before (16, 24, or 32 bytes).

14.5.6.2 AESU Key Size Register (AESUKSR)

The AESU key size register stores the number of bytes in the key (16,24,32). Any key data beyond the number of bytes in the key size register will be ignored. AESUKSR is cleared when the AESU is reset or re-initialized. If a key size other than 16, 24, or 32 bytes is specified, an illegal key size error will be generated. If the key size register is modified during processing, a context error will be generated.

NOTE

Writing to this register while in debug mode generates an illegal size error. Disable the illegal size error in the associated Interrupt Control Register prior to performing debug operations.

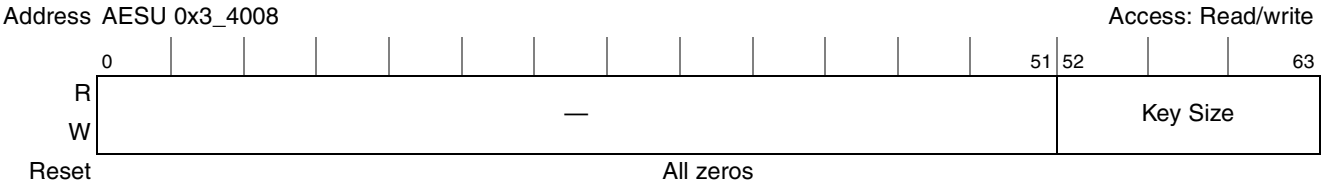


Figure 14-52. AESU Key Size Register

14.5.6.3 AESU Data Size Register (AESUDSR)

AESUDSR is used to advise the AESU of the size of the data to be processed. Depending on the AES mode selected, data size must be divisible by a specific block size or a data size error will occur. In ECB, CBC, and CTR mode, the AESU does not automatically pad messages out to 128-bit blocks, therefore when operating in ECB, CBC, or CTR mode, the message processed by the AESU must be divisible by 128-bits or a data size error will occur. When operating in CCM mode, data size must be divisible by 8-bits or a data size error will occur. In XOR mode the data size must be a multiple of 256 bits (32 bytes). If an improper data size is written, a data size error is generated. Only the lowest 3, 7, or 8 bits of the data size register are checked to determine if there is a data size error. Since all upper bits are ignored, the entire message length (in bits) can be written to this register.

In normal operation, the full message length to be encrypted or decrypted with the AESU is copied from the descriptor to the AESU data size register, however only bits 56:63 are checked to determine if there is a data size error. If 57:63 are all zeroes, the message is evenly divisible into 128-bit blocks.

This register is cleared when the AESU is reset or re-initialized. If a data size other than 128-bits is specified, an illegal data size error will be generated. Writing to this register signals the AESU to start processing data from the input FIFO as soon as it is available. If the value of data size is modified during processing, a context error will be generated.

NOTE

Writing to this register while in debug mode generates an illegal size error. Disable the illegal size error in the associated Interrupt Control Register prior to performing debug operations.

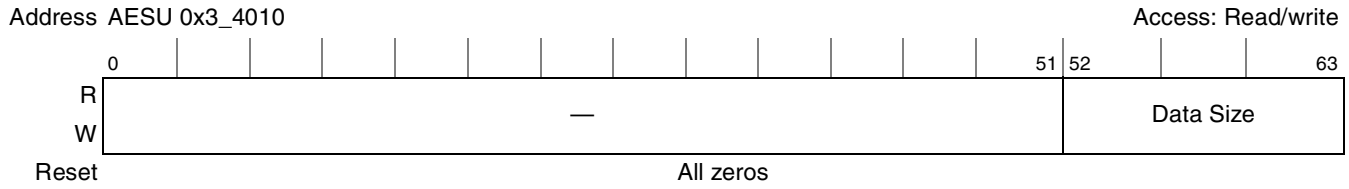


Figure 14-53. AESU Data Size Register

14.5.6.4 AESU Reset Control Register (AESURCR)

AESURCR allows three levels reset of just AESU, as defined by the three self-clearing bits:

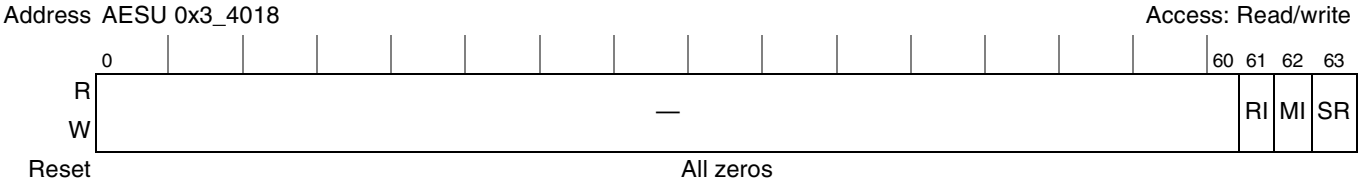


Figure 14-54. AESU Reset Control Register

Table 14-17 describes AESU reset control register fields.

Table 14-41. AESU Reset Control Register Field Descriptions

Bits	Names	Description
0–60	—	Reserved
61	RI	Reset Interrupt. Writing this bit active high causes AESU interrupts signalling DONE and ERROR to be reset. It further resets the state of the AESU interrupt status register. 0 Don't reset 1 Reset interrupt logic
62	MI	Module initialization is nearly the same as software reset, except that the interrupt control register remains unchanged. This module initialization includes execution of an initialization routine, completion of which is indicated by the RESET_DONE bit in the AESU status register 0 Don't reset 1 Reset most of AESU
63	SR	Software reset is functionally equivalent to hardware reset (the RESET# pin), but only for AESU. All registers and internal state are returned to their defined reset state. Upon negation of SW_RESET, the AESU will enter a routine to perform proper initialization of the parameter memories. The RESET_DONE bit in the AESU status register will indicate when this initialization routine is complete 0 Don't reset 1 Full AESU reset

14.5.6.5 AESU Status Register (AESUSR)

AESUSR, shown in Figure 14-55, reflects the state of six status outputs. Writing to this location causes an address error being reflected in the AESU interrupt status register.

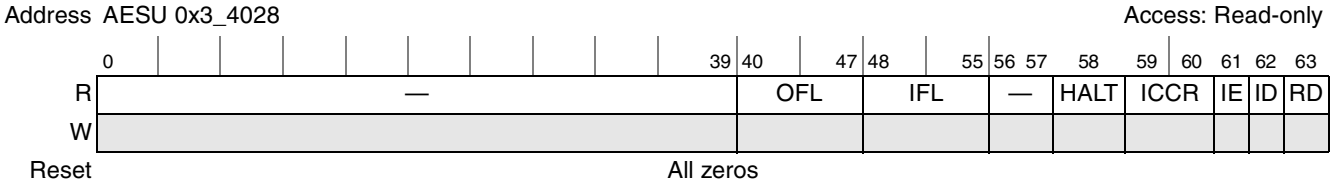


Figure 14-55. AESU Status Register

Table 14-23 describes AESU status register fields.

Table 14-42. AESU Status Register Field Descriptions

Bits	Name	Description
0–39	—	Reserved
40–47	OFL	The number of dwords currently in the output FIFO
48–55	IFL	The number of dwords currently in the input FIFO
56–57	—	Reserved
58	HALT	Halt. Indicates that the AESU has halted due to an error. 0 AESU not halted 1 AESU halted **Note:** Because the error causing the AESU to stop operating may be masked to the interrupt status register, the status register is used to provide a second source of information regarding errors preventing normal operation.
59–60	ICCR	Integrity check comparison result 00 No integrity check comparison was performed. 01 The integrity check comparison passed. 10 The integrity check comparison failed. 11 Reserved A passed or failed result is generated only if ICV checking is enabled and the cipher mode selected is CCM with ICV comparison
61	IE	Interrupt Error.This status bit reflects the state of the ERROR interrupt signal, as sampled by the controller interrupt status register (Section 14.7.2.2, “Interrupt Status Register (ISR)”). 0 AESU is not signaling error 1 AESU is signaling error
62	ID	Interrupt Done. This status bit reflects the state of the DONE interrupt signal, as sampled by the Controller Interrupt Status Register (Section 14.7.2.2, “Interrupt Status Register (ISR)”). 0 AESU is not signaling done 1 AESU is signaling done
63	RD	Reset Done. This status bit, when high, indicates that AESU has completed its internal reset sequence. 0 Reset in progress 1 Reset done

14.5.6.6 AESU Interrupt Status Register (AESUISR)

The AESU interrupt status register tracks the state of possible errors, if those errors are not masked, via the AESU interrupt control register. The definition of each bit in the interrupt status register is shown in Figure 14-56.

Address AESU 0x3_4030 Access: Read-only

	0–48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
R	—	ICE	—	IE	ERE	CE	KSE	DSE	ME	AE	OFE	IFE	—	IFO	OFU	—
W																
Reset	All zeros															

Figure 14-56. AESU Interrupt Status Register

Table 14-19 describes AESU interrupt register fields.

Table 14-43. AESU Interrupt Status Register Field Descriptions

Bits	Name	Description
0–48	—	Reserved
49	ICE	Integrity check error 0 No error detected 1 Integrity check error detected. An ICV check was performed and the supplied ICV did not match the one computed by the AESU.
50	—	Reserved
51	IE	Internal Error. An internal processing error was detected while the AESU was processing. 0 No error detected 1 Internal error **Note:** This bit will be asserted any time an enabled error condition occurs and can only be cleared by setting the corresponding bit in the Interrupt Control Register or by resetting the AESU.
52	ERE	Early Read Error. The AESU IV Register was read while the AESU was processing. 0 No error detected 1 Early read error
53	CE	Context Error. An AESU Key Register, the Key Size Register, Data Size Register, Mode Register, or IV Register was modified while AESU was processing 0 No error detected 1 Context error
54	KSE	Key Size Error. An inappropriate value (not 16, 24 or 32bytes) was written to the AESU Key Size Register 0 No error detected 1 Key size error
55	DSE	Data Size Error (DSE): A value was written to the AESU Data Size Register that is not a multiple of 128 bits. 0 No error detected 1 Data size error
56	ME	Mode Error. Indicates that invalid data was written to a register or a reserved mode bit was set. 0 Valid Data 1 Reserved or invalid mode selected
57	AE	Address Error. An illegal read or write address was detected within the AESU address space. 0 No error detected 1 Address error

Table 14-43. AESU Interrupt Status Register Field Descriptions (continued)

Bits	Name	Description
58	OFE	Output FIFO Error. The AESU output FIFO was detected non-empty upon write of AESU data size register. 0 No error detected 1 Output FIFO non-empty error
59	IFE	Input FIFO Error. The AESU input FIFO was detected non-empty upon generation of DONE interrupt. 0 No error detected 1 Input FIFO non-empty error
60	—	Reserved
61	IFO	Input FIFO Overflow. The AESU Input FIFO has been pushed while full. 0 No error detected 1 Input FIFO has overflowed **Note:** When operating as a master, the implements flow-control, and FIFO size is not a limit to data input. When operated as a target, the cannot accept FIFO inputs larger than 512 Bytes without overflowing.
62	OFU	Output FIFO Underflow. The AESU Output FIFO has been read while empty. 0 No error detected 1 Output FIFO has underflow error
63	—	Reserved

14.5.6.7 AESU Interrupt Control Register (AESUICR)

The AESU interrupt control register, shown in Figure 14-57, controls the result of detected errors. For a given error (as defined in Section 14.5.6.6, “AESU Interrupt Status Register (AESUISR)”), if the corresponding bit in this register is set, then the error is ignored; no error interrupt occurs and the interrupt status register is not updated to reflect the error. If the corresponding bit is not set, then upon detection of an error, the interrupt status register is updated to reflect the error, causing assertion of the error interrupt signal, and causing the module to halt processing.

Address AESU 0x3_4038 Access: Read/write

	0–‘47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
R W	—		ICE	—	IE	ERE	CE	KSE	DSE	ME	AE	OFE	IFE	—	IFO	OFU	—
Reset	All zeros	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-57. AESU Interrupt Control Register

Table 14-44 describes the AESU interrupt control register fields.

Table 14-44. AESU Interrupt Control Register Field Descriptions

Bits	Name	Description
0–48	—	Reserved
49	ICE	Integrity check error. The supplied ICV did not match the one computed by the AESU. 0 Integrity check error enabled 1 Integrity check error disabled
50	—	Reserved

Table 14-44. AESU Interrupt Control Register Field Descriptions (continued)

Bits	Name	Description
51	IE	Internal Error. An internal processing error was detected while the AESU was processing. 0 Internal error enabled 1 Internal error disabled
52	ERE	Early Read Error. The AESU IV Register was read while the AESU was processing. 0 Early read error enabled 1 Early read error disabled
53	CE	Context Error. An AESU Key Register, the Key Size Register, Data Size Register, Mode Register, or IV Register was modified while the AESU was processing. 0 Context error enabled 1 Context error disabled
54	KSE	Key Size Error. An inappropriate value (non 16, 24 or 32 bytes) was written to the AESU key size register 0 Key size error enabled 1 Key size error disabled
55	DSE	Data Size Error. Indicates that the number of bits to process is out of range. 0 Data size error enabled 1 Data size error disabled
56	ME	Mode Error. Indicates that invalid data was written to a register or a reserved mode bit was set. 0 Mode error enabled 1 Mode error disabled
57	AE	Address Error. An illegal read or write address was detected within the AESU address space. 1 Address error disabled 0 Address error enabled
58	OFE	Output FIFO Error. The AESU Output FIFO was detected non-empty upon write of AESU data size register 0 Output FIFO non-empty error enabled 1 Output FIFO non-empty error disabled
59	IFE	Input FIFO Error. The AESU Input FIFO was detected non-empty upon generation of DONE interrupt 0 Input FIFO non-empty error enabled 1 Input FIFO non-empty error disabled
60	—	Reserved
61	IFO	Input FIFO Overflow. The AESU Input FIFO has been pushed while full. 0 Input FIFO overflow error enabled 1 Input FIFO overflow error disabled
62	OFU	Output FIFO Underflow The AESU Output FIFO has been read while empty. 0 Output FIFO underflow error enabled 1 Output FIFO underflow error disabled
63	—	Reserved

14.5.6.8 AESU End of Message Register (AESUEMR)

AESUEMR, shown in Figure 14-58, is used to indicate an AES operation may be completed. After the final message block is written to the input FIFO, the end of message register must be written. The value in the data size register will be used to determine how many bits of the final message block (always 128) will be processed. Writing to this register causes the AESU to process the final block of a message, allowing it

to signal DONE. A read of this register always returns a zero. AESUEMR is used only when the is operated as a slave. The descriptors and crypto-channel activate the AESU (via an internally generated write to the end of message register) when the SEC acts as an initiator.

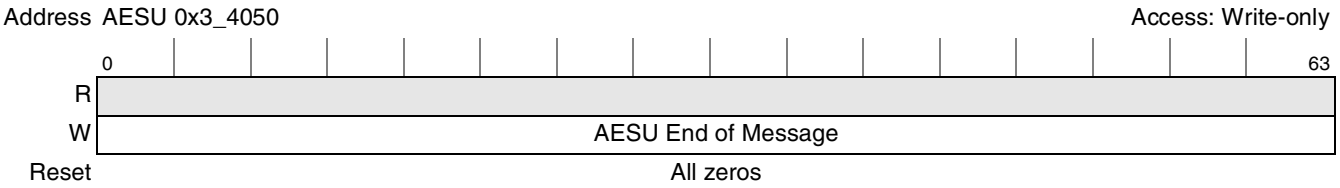


Figure 14-58. AESU End of Message Register

14.5.6.9 AESU Context Registers

There are three 64-bit context data registers that allow the host to read/write the contents of the context used to process the message. The context must be written prior to the key data. If the context registers are written during message processing, a context error will be generated. All context registers are cleared when a hard/soft reset or initialization is performed.

The context registers must be read when changing context and restored to their original values to resume processing an interrupted message (CBC, CTR and CCM modes). For CTR and CCM mode, all seven 64-bit context registers must be read to retrieve context, and all seven must be written back to restore context. Effectively, the user must read the four empty ‘place holder’ context registers in addition to the three context registers holding the Counter and Counter Modulus when in CTR mode. The contents of the ‘empty’ context registers need not be preserved, but when restoring the CTR mode context, the ‘empty’ registers must be filled with 32 bytes of zeros before writing the saved Counter and Counter Modulus.

Context should be loaded with the lower bytes in the lowest 64-bit context register. The context registers are summarized in Figure 14-59.

Context Register (64-bits each)

Cipher Mode	1	2	3	4	5	6	7
ECB	—	—	—	—	—	—	—
CBC	IV1[1]	IV2[1]	—	—	—	—	—
CTR	—	—	—	—	Counter [1]		Counter Modulus[1]
SRT	Counter [1]		Counter Modulus[1]	—	—	—	—
CCM	IV[1] / MAC Tag		Encrypted MAC[2]/Decrypted MAC/Encrypted Counter		Counter [1]		Counter Modulus[1,3]

1 Must be written at the start of a new message
2 Must be written at start of new CCM decryption
3 Header size/MAC size is only used if AES-CCM processing is suspended and resumed.

Figure 14-59. AESU Context Register

14.5.6.9.1 Context for CBC Mode

Within the Context register, for use in CBC mode, are two 64-bit context data registers that allow the host to read/write the contents of the initialization vector (IV):

- IV1 holds the least significant bytes of the initialization vector (bytes 1–8).
- IV2 holds the most significant bytes of the initialization vector (bytes 9–16).

The IV must be written prior to the message data. If the IV registers are written during message processing, or the CBC mode bit is not set, a context error will be generated.

The IV registers may only be read after processing has completed, as indicated by the assertion of Interrupt Done DONE in the AESU status register as shown in Section 14.5.6.5, "AESU Status Register (AESUSR)." If the IV registers are read prior to assertion of Interrupt Done, an early read error will be generated.

The IV registers must be read when changing context and restored to resume processing an interrupted message (CBC mode only).

14.5.6.9.2 Context for Counter Mode

In counter mode, a random 128-bit initial counter value is incremented modulo 2^n with each block processed. The modulus size can be set between 2^8 through 2^{128}, by powers of 8. The running counter is encrypted and eXclusive-ORed with the plaintext to derive the ciphertext, or with the ciphertext to recover the plaintext.

In CTR mode, the block counter is incremented modulo 2^M. The value of M is specified by writing to Context Register 3 as described in Table 14-45.

Value Written	Modulus	Value Written	Modulus
8	2^8	72	2^{72}
16	2^{16}	80	2^{80}
24	2^{24}	88	2^{88}
32	2^{32}	96	2^{96}
40	2^{40}	104	2^{104}
48	2^{48}	112	2^{112}
56	2^{56}	120	2^{120}
64	2^{64}	128	2^{128}

Table 14-45. Counter Modulus

14.5.6.9.3 Context for SRT Mode

As was noted in the AESU mode register, SRT is not a new AES mode, it is an AESU method of performing AES-CTR mode with reduced context loading overhead specifically for performing SRTP. It should be used with descriptor type 0010_0'srtp'. As with counter mode, a random 128-bit initial counter value is incremented modulo 2^n with each block processed. The modulus size can be set between 2^8 through 2^{128}, by powers of 8. The running counter is encrypted and eXclusive-ORed with the plaintext to derive the

ciphertext, or with the ciphertext to recover the plaintext. The block counter is incremented modulo 2^M. The value of M is specified by writing to Context Register 3 as described in Table 14-45.

The only difference between SRT mode and CTR mode is in SRT mode, the AES Context is loaded and read via Context Registers 1–3, with no requirement to access Context Registers 4–7. In CTR mode, Context Registers 1–4 must be loaded with zeros, with the Counter and Modulus being loaded into and read from Context Registers 5–7.

14.5.6.9.4 Context for CCM Mode

The SEC AESU is capable of performing single pass encryption and MAC generation. The host is required to order the CCM context is such a way that the context can be fetched as a contiguous string into the context registers, prior to encryption/MAC generation or decryption/MAC validation. A complete explanation of the context and ordering can be found below.

The context for CCM encryption/MAC generation is:

Reg 1–2 Session specific128 bit Initialization Vector (from memory)

Reg 3–4 128 bits of zero padding

Reg 5–6 Session Specific Counter (Initial Counter Value) (from memory)

Reg 7 Counter Modulus Should be fixed at 0x0000_0080.

Note: The counter modulus for CCM mode is currently defined as 2^128. This value has been made programmable in the SEC in case the final version of 802.11i uses a different counter modulus. Because this is a programmable field, it must be generated and stored along with other session specific information for loading into the AESU context register prior to CCM encryption.

CCM encryption processing

With the session specific key and context, the AESU will perform the following operations.

1. Initialize the IV, and encrypt with the symetric key.
2. In CBC fashion, take the output of step 1, hash with the first block of plaintext, and encrypt with the symetric key.
3. Continue as in step 2 until the final block of plaintext has been processed. The result of the encryption of the final block of plaintext with the symetric key is the MAC Tag. The full 128bits of MAC data is written to context registers 1–2, for use in the next phase of CCM processing. Once the MAC Tag has been generated (step 3), the MAC tag, along with the plaintext is encrypted with the AESU operating in Counter mode.
4. The first item to be encrypted in counter mode is the Counter (Initial Counter Value) from Context Registers 5–6. The counter is encrypted with the symmetric key, and the result is hashed with the MAC Tag (retrieved from Context Reg 1–2) to produce the Encrypted MAC, which is then stored in Context Registers 3–4. At the completion of CCM encrypt processing, this Encrypted MAC is output to memory (per the descriptor pointer) for the host to append to the 802.11i frame. Note: The Encrypted MAC written out to memory by the AESU is the full 128-bits. The host must only append the most significant 64-bits to the frame as the encrypted MAC.

5. The counter value is incremented, and is then encrypted with the symmetric key. The result is then hashed with the first block of plaintext to produce the first block of cipher text. The ciphertext is placed in the AESU output FIFO.
6. The counter continues to be incremented, and encrypted with the symmetric key, with the result hashed with each successive block of plaintext, until all plaintext has been converted to ciphertext. The SEC controller will manage FIFO reads and writes, fetching plaintext and writing ciphertext per the pointers provided in the descriptor. When all ciphertext and the encrypted MAC has been output, the CCM encrypt operation is complete.

The context for CCM decryption/MAC generation is as follows:

Reg 1–2 Session specific128 bit Initialization Vector (from memory)

Reg 3–4 Encrypted MAC (from received frame) + 64 bits of zero padding

Reg 5–6 Session Specific Counter (Initial Counter Value) (from memory)

Reg 7 Counter Modulus Should be fixed at 0x0000_0080.

NOTE

The counter modulus for CCM mode is currently defined as 2^128. This value has been made programmable in the SEC to in case the final version of 802.11i uses a different counter modulus. Because this is a programmable field, it must be generated and stored along with other session specific information for loading into the AESU Context Register prior to CCM decryption.

CCM decryption processing is the reverse of encryption;

With the session specific key and context, the AESU will perform the following operations.

1. Initialize the IV, and encrypt with the symmetric key. Simultaneously, the counter (Initial Counter Value) from Context Registers 5–6 is encrypted with the symmetric key. The result is hashed with the Encrypted MAC (from Context Register 3–4), and the resulting Original MAC is written to Context Reg 3–4, overwriting the Encrypted MAC.
   Note: Strictly speaking, the Counter is encrypted with the symmetric key, however the AESU should be set for ‘decrypt’ to perform the counter and CBC processes in the correct order.
2. The 802.11 frame header is hashed with the encrypted IV. (The AESU automatically determines the header length.) Simultaneously, the counter is incremented, and is then encrypted with the symmetric key. The result is then hashed with the first block of ciphertext to produce the first block of plaintext. The plaintext is placed in the AESU output FIFO, while simultaneously, in CBC fashion, a copy of the first block of plaintext is hashed with the output of encryption of the 802.11 frame header. The output is encrypted with the symmetric key.
3. As each ciphertext block is converted to plaintext, the plaintext is CBC encrypted. When the final plaintext block has been processed, the CBC MAC (MAC Tag) is written to Context Registers 1–2. The first 64 bits of the MAC Tag are compared to the MAC Tag recovered in step 1.

NOTE

For both encrypt and decrypt operations, if the 802.11 frame is being processed as a whole (not split across multiple descriptors), the ‘Initialize’ and ‘Final MAC’ bits should be set in the AESU Mode Register.

14.5.6.9.5 AESU Key Registers

The AESU key registers hold from 16, 24, or 32 bytes of key data, with the first 8 bytes of key data written to key 1. Any key data written to bytes beyond the value written to the key size register will be ignored. The key data registers are cleared when the AESU is reset or re-initialized. If these registers are modified during message processing, a context error will be generated.

The key data registers may be read when changing context in decrypt mode. To resume processing, the value read must be written back to the key registers and the ‘restore decrypt key’ bit must be set in the mode register. This eliminates the overhead of expanding the key prior to starting decryption when switching context.

14.5.6.9.6 AESU FIFOs

The AESU fetches data 128 bits at a time from the input FIFO. During processing, the input data is encrypted or decrypted with the key and initialization vector (CBC mode only) and the results are placed in the output FIFO. The output size is the same as the input size.

Writing to the FIFO address space places 64 bits of message data into the input FIFO. The input FIFO may be written any time the IFW signal is asserted (as indicated in the AESU status register). This will indicate that the number of bytes of available space is at or above the threshold specified in the mode register. There is no limit on the total number of bytes in a message. The number of bits in the final message block must be set in the data size register.

Reading from the FIFO address space will pop 64 bits of message data from the output FIFO. The output FIFO may be read any time the OFR signal is asserted (as indicated in the AESU status register). This will indicate that the number of bytes in the output FIFO is at or above the threshold specified in the mode register.

14.6 Crypto-Channels

A channel in the SEC manages the execution of each cryptographic task, making use of one or more of the SEC’s execution units (EUs). Control information and data pointers for a given task are stored in the form of a descriptor (see Section 14.4.1, “Descriptor Structure”) in system memory or in the channel itself. A descriptor determines what EUs will be used, how they will be configured, where to fetch needed data, and where to store the results. To invoke cryptographic tasks, the host constructs a descriptor, selects a channel, and writes a pointer to the descriptor into the selected channel’s Fetch FIFO. Operations performed by channels include the following (not necessarily in this order):

- If the channel is idle and its fetch FIFO is non-empty, read the next descriptor pointer from the Fetch FIFO, and use this pointer to read the descriptor into the channel’s descriptor buffer.
- Request from the controller the assignment of one or more EUs for the exclusive use of the channel. Where necessary, configure the secondary EU to snoop input or output data intended for the primary EU.

- Upon notification of completion of the EU reset sequence, initialize mode registers in the assigned EU.
- Initialize EUs and write to EU registers such as key size and text-data size.
- Transfer data parcels (up to 32Kbytes) from system memory into assigned EU input registers and FIFOs. This may involve using link tables to gather input data that has been split into multiple segments which are stored in various locations of system memory.
- Transfer data parcels (up to 32Kbytes) from assigned EU output registers and FIFOs to system memory space. This may involve using link tables to scatter output data into multiple segments which are stored in various locations of system memory. For the RAID-XOR descriptor type, the channel rotates among three data sources, fetching 32 bytes from each source.
- Initialize the EU-Go register (where applicable) in the assigned EU upon completion of last EU write indicated by the descriptor. The channel will wait for a indication from the EU that processing of input text-data is complete before proceeding with further activity after writing EU-Go.
- Reset assigned EU(s).
- Release assigned EU(s).
- When a descriptor has been completely processed, provide feedback to the host, in the form of interrupt and/or descriptor header write-back to system memory.
- When descriptor processing is halted due to an error, provide feedback to the host via interrupt.

The channel will wait indefinitely for the controller to complete a requested activity before continuing to the next step of descriptor processing.

14.6.1 Crypto-Channel Registers

Crypto-channel registers are described in the following sections.

14.6.1.1 Crypto-Channel Configuration Register (CCCR)

CCCR contains five operational bits permitting configuration of the crypto-channel as shown in Figure 14-60. Table 14-46 describes the CCCR.

Address Channel_1 0x1108
Channel_2 0x1208
Channel_3 0x1308
Channel_4 0x1408

Access: User read/write

	0 – 29	30	31
R	—	CON	R
W			
Reset	All zeros		

	32 – 54	55	56	57	58	59	60	61	62	63
R	—	BS	IWSE	AWSE	EAE	CDWE	—	NT	CDIE	—
W										
Reset	All zeros									

Figure 14-60. Crypto-Channel Configuration Register (CCCR)

Table 14-46. CCCR Field Descriptions

Bits	Name	Description
0–29	—	Reserved, set to zero
30	CON	Continue bit 0 No special action. 1 Causes the same channel reset actions as bit R, except that the fetch FIFO and the lower half of the CCR register are not cleared. After the reset sequence is complete, this bit automatically returns to 0 and the channel resumes normal operation, servicing the next descriptor pointer in the fetch FIFO, if any.
31	R	Reset channel 0 No special action. 1 Causes a software reset of the channel, clearing all its internal state. The details of the software reset actions depend upon what the channel is doing when the bit is set: If the R bit is set while the channel is requesting an EU assignment from the controller, the channel cancels its request by asserting the release output signals. The channel then resets all its registers, clears the R bit, and return the channel state machine to the idle state. If the R bit is set after the channel has been assigned an EU, the channel requests a write from the controller to set the software reset bit of the EU. If a secondary EU has been reserved, the channel requests a write to reset that EU as well. The channel next asserts the appropriate release signal to notify the controller that the channel has finished with the reserved EU(s). The channel then resets all the registers, clears the RESET bit and returns the channel state machine to the idle state.
32–54	—	Reserved, set to zero
55	BS	Burst size— The SEC accesses long text-data parcels in main memory through bursts of programmable size: 0 Burst size is 64 bytes 1 Burst size is 128 bytes
56	IWSE	ICV writeback status enable 0 No special action. 1 If the descriptor calls for ICV comparison, then at the completion of descriptor processing, write back the status of all EUs into the header dword.
57	AWSE	Always writeback status enable 0 No special action. 1 At the completion of processing each descriptor, write back the status of all EUs into the header dword. In this case, IWSE has no effect.
59	CDWE	Channel done writeback enable 0 Channel done writeback disabled. 1 Channel done writeback enabled. Upon completion of descriptor processing, if the NT bit is set for Global, or if the DN (Done Notification) bit is set in the header word of the descriptor, notify the host by writing back the descriptor header with the writeback information shown in Figure 14-61. This enables the host to poll the memory location of the original descriptor header to determine if that descriptor has been completed.
60	—	Reserved, set to zero
61	NT	Notification type. This bit controls when the channel will generate channel done notification. Channel done notification can take the form of an interrupt or modified header writeback or both, depending on the state of the CDIE and CDWE control bits. 0 Global notification. The channel will generate channel done notification (if enabled) at the end of each descriptor. 1 Selected notification. The channel will generate channel done notification (if enabled) at the end of every descriptor with the DONE bit set in the descriptor header.

Table 14-46. CCCR Field Descriptions (continued)

Bits	Name	Description
62	CDIE	Channel done interrupt enable 0 Channel done interrupt disabled 1 Channel Done Interrupt enabled. Upon completion of descriptor processing, if the NT bit is set for Global, or if the DN (Done Notification) bit is set in the header word of the descriptor, then notify the host by asserting an interrupt. Refer to Section 14.6.2, "Interrupts," for complete description of channel interrupt operation.
63	—	Reserved, set to zero

Figure 14-61 shows the format of the header dword when the channel is configured to perform done notification via header writeback. A detailed description of the header dword fields used in writeback notification is provided in Table 14-47.

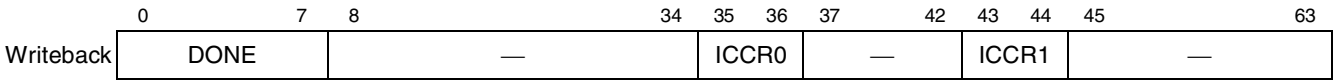


Figure 14-61. Header Dword Writeback Format

Table 14-47. Header Dword Writeback Field Descriptions

Bits	Name	Description
0–7	DONE	When done writeback is used, then at the completion of descriptor processing this byte is written with the value 0xFF. To determine when done writeback is used, see the CDWE, NT, and CDIE fields in the channel configuration register (see Table 14-47).
8–34	—	Reserved.
35–36	ICCR0	Integrity check comparison result from primary. These bits are supplied by the primary EU when descriptor processing is complete. 00 No integrity check comparison was performed. 01 The integrity check comparison passed. 10 The integrity check comparison failed. 11 Reserved
37–42	—	Reserved.
43–44	ICCR1	Integrity check comparison result from secondary. These bits are supplied by the secondary EU (if any) when descriptor processing is complete. 00 No integrity check comparison was performed. 01 The integrity check comparison passed. 10 The integrity check comparison failed. 11 Reserved
45–63	—	Reserved.

14.6.1.2 Crypto-Channel Pointer Status Register (CCPSR)

CCPSR contains status fields and counters which provide the user with status information regarding the channel's actual processing of a given descriptor.

Address Channel_1 0x01110
Channel_2 0x01210
Channel_3 0x01310
Channel_4 0x01410

Access: Read-only

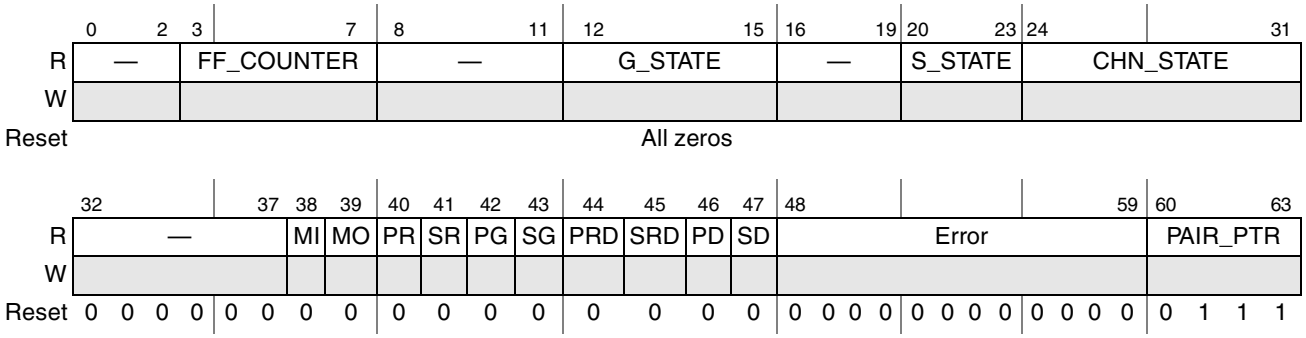


Figure 14-62. Crypto-Channel Pointer Status Register

Table 14-48 describes the crypto-channel pointer status register fields.

Table 14-48. Crypto-Channel Pointer Status Register Signals

Bits	Name	Description
0–2	—	Reserved.
3–7	FF_COUNTER	Fetch FIFO Counter. The fetch FIFO can store up to 24 pointers to descriptors in system memory. This 5 bit counter indicates how many fetch pointers are currently stored in the fifo.
8–11	—	Reserved.
12–15	G_STATE	Gather state machine state. Indicates which stage the crypto-channel is while performing the gather function. Table 14-49 shows the meaning of all possible values of G_STATE. **Note:** G_state is documented for information only. The user typically does not care about the gather state machine.
16–19	—	Reserved.
20–23	S_STATE	Scatter state machine state. Indicates which stage the crypto-channel is while performing the scatter function. Table 14-50 shows the meaning of all possible values of S_STATE. **Note:** S_state is documented for information only. The user typically does not care about the scatter state machine.
24–32	CHN_STATE	Crypto-channel state machine. Indicates the stage of the crypto-channel in the sequence of fetching and processing data descriptors. Table 14-49 shows the meaning of all possible values of STATE. **Note:** CHN_State is documented for information only. The user typically does not care about the crypto-channel state machine.
31–37	—	Reserved, set to zero
38	MI	Multi EU input. Reflects the type of snooping the channel will perform, as programmed by the 'Snoop Type' bit in the descriptor header. 0 Data input snooping by secondary EU disabled. 1 Data input snooping by secondary EU enabled.

Table 14-48. Crypto-Channel Pointer Status Register Signals (continued)

Bits	Name	Description
39	MO	Multi EU output. Reflects the type of snooping the channel will perform, as programmed by the 'Snoop Type' bit in the descriptor header. 0 Data output snooping by secondary EU disabled. 1 Data output snooping by secondary EU enabled.
40	PR	Request primary EU assignment. The PRI_REQ bit is set when descriptor processing is initiated in dynamic mode and the Op_0 field in the descriptor header contains a valid EU identifier. This bit is cleared when the request is granted, which will be reflected in the status register by the setting the PRI_GRANT bit. 0 Primary EU Assignment Request is inactive. 1 The crypto-channel is requesting assignment of primary EU to the channel. The channel will assert the EU request signal indicated by the op0 field in the Descriptor Header register as long as this bit remains set.
41	SR	Request secondary EU assignment. The SEC_REQ bit is set when descriptor processing is initiated in dynamic mode and the Op_1 field in the descriptor header contains a valid EU identifier. This bit is cleared when the request is granted, which will be reflected in the status register by the setting the SEC_GRANT bit. 0 Secondary EU Assignment Request is inactive. 1 The crypto-channel is requesting assignment of secondary EU to the channel. The channel will assert the EU request signal indicated by the Op_1 field in the descriptor header register as long as this bit remains set.
42	PG	Primary EU granted. The PRI_GRANT bit reflects the state of the EU grant signal for the requested primary EU from the controller. 0 The primary EU grant signal is inactive. 1 The EU grant signal is active indicating the controller has assigned the requested primary EU to the channel.
43	SG	Secondary EU granted. The SEC_GRANT bit reflects the state of the EU grant signal for the requested secondary EU from the controller. 0 The secondary EU grant signal is inactive. 1 The EU grant signal is active indicating the controller has assigned the requested secondary EU to the channel.
44	PRD	Primary EU reset done. The PRI_RST_DONE bit reflects the state of the reset done signal from the assigned primary EU. 0 The assigned primary EU reset done signal is inactive. 1 The assigned primary EU reset done signal is active indicating its reset sequence has completed and it is ready to accept data.
45	SRD	Secondary EU reset done. The SEC_RST_DONE bit reflects the state of the reset done signal from the assigned secondary EU. 0 The assigned secondary EU reset done signal is inactive. 1 The assigned secondary EU reset done signal is active indicating its reset sequence has completed and it is ready to accept data.
46	PD	Primary EU done. The PRI_DONE bit reflects the state of the DONE interrupt from the assigned primary EU. 0 The assigned primary EU done interrupt is inactive. 1 The assigned primary EU done interrupt is active indicating the EU has completed processing and is ready to provide output data.

Table 14-48. Crypto-Channel Pointer Status Register Signals (continued)

Bits	Name	Description
47	SD	Secondary EU done. The SEC_DONE bit reflects the state of the DONE interrupt from the assigned secondary EU. 0 The assigned secondary EU done interrupt is inactive. 1 The assigned secondary EU done interrupt is active indicating the EU has completed processing and is ready to provide output data.
48–59	ERROR	Crypto-channel error status. This field reflects the error status of the crypto-channel. When a channel error interrupt is generated, this field will reflect the source of the error. The bits in the ERROR field are registered at specific stages in the descriptor processing flow. Once registered, an error can only be cleared only by resetting the crypto-channel or writing the appropriate registers to initiate the processing of a new descriptor. Table 14-52 lists the conditions which can cause a crypto-channel error and how they are represented in the ERROR field.
60–63	PAIR_PTR	Descriptor buffer register length/pointer pair. This field indicates which of the length/pointer pairs are currently being processed by the channel. Table 14-53 shows the meaning of all possible values of the PAIR_PTR field.

Table 14-49 shows the values of the gather state machine.

Table 14-49. G_STATE Field Values

Value	Gather State Machine	Value	Gather State Machine
0x0	idle	0x8	request_bytes_data_tarns_done
0x1	load_4pointers_frm_gather_table	0x9	increase_table_pointer
0x2	load_4pointers_frm_gather_table_done	0xA	update_gather_pointer
0x3	next_bit_set_load_next_gather_table	0xB	gather_table_done
0x4	process_gather_pointer	0xC	gather_error
0x5	request_block_data_trans	0xD	load_next_4pointers_frm_gather_table
0x6	request_block_data_trans_done	0xE–0xF	reserved
0x7	request_bytes_data_tarns	—	—

Table 14-50 shows the values of the scatter state machine.

Table 14-50. S_STATE Field Values

Value	Scatter State Machine:
0x0	idle
0x1	load_4pointers_frm_scatter_table
0x2	load_4pointers_frm_scatter_table_done
0x3	next_bit_set_load_next_scatter_table
0x4	process_scatter_pointer
0x5	request_block_data_trans
0x6	request_block_data_trans_done
0x7	request_bytes_data_tarns

Table 14-50. S_STATE Field Values (continued)

Value	Scatter State Machine:
0x8	request_bytes_data_tarns_done
0x9	increase_table_pointer
0xA	update_scatter_pointer
0xB	scatter_table_done
0xC	scatter_error
0xD	load_next_4pointers_frm_scatter_table
0xE–0xF	reserved

Table 14-51 shows the values of crypto-channel states.

Table 14-51. CHN_STATE Field Values

Value	Channel State	Value	Channel State
0x00	IDLE	0x22	EVALUATE_RESET
0x01	PROCESS_HEADER	0x23	RESET_WRITE_RESET_PRI
0x02	FETCH_DESCRIPTOR	0x24	RESET_RELEASE_PRI_CHA
0x03	CHANNEL_DONE	0x25	RESET_WRITE_RESET_SEC
0x04	CHANNEL_DONE_IRQ	0x26	RESET_RELEASE_SEC_CHA
0x05	CHANNEL_DONE_WRITEBACK	0x27	RESET_CHANNEL
0x06	CHANNEL_DONE_NOTIFICATION	0x28	WRITE_DATASIZE_PRI_POST
0x07	CHANNEL_ERROR	0x29	RESET_RELEASE_ALL
0x08	REQUEST_PRI_CHA	0x2A	RESET_RELEASE_ALL_DELAY
0x09	INC_DATA_PAIR_POINTER	0x2B	REQUEST_SEC_CHA
0x0A	DELAY_DATA_PAIR_UPDATE	0x2C	WRITE_DATASIZE_SEC
0x0B	EVALUATE_DATA_PAIRS	0x2D	WRITE_ICV_SIZE
0x0C	WRITE_RESET_PRI	0x2E	WRITE_SEC_CHA_GO_SNOOPOUT
0x0D	RELEASE_PRI_CHA	0x2F	WRITE_PRI_CHA_GO_SNOOPIN
0x0E	WRITE_RESET_SEC	0x30	WRITE_SEC_CHA_GO_SNOOPIN
0x0F	RELEASE_SEC_CHA	0x31	DELAY_1CYCLE
0x10	PROCESS_DATA_PAIRS	0x33	TRANS_EXTENT_READ
0x11	WRITE_MODE_PRI	0x34	TRANS_EXTENT3
0x12	WRITE_MODE_SEC	0x35	TRANS_EXTENT4
0x13	WRITE_DATASIZE_PRI	0x36	XOR_WRITE_READ_REG
0x14	DELAY_RNGA_DONE	0x37	DELAY_SEC_DONE_TLS
0x15	WRITE_DATASIZE_SEC_SNOOPIN	0x38	MAC_TO_CIPHER
0x16	TRANS_REQUEST_WRITE_SNOOPIN	0x39	MAC_TO_CIPHER_DONE
0x17	DELAY_PRI_SEC_DONE	0x3A	READ_PRI_STATUS
0x18	TRANS_REQUEST_WRITE	0x3C	READ_SEC_STATUS
0x19	WRITE_KEY_SIZE	Others	Reserved
0x1B	DELAY_PRI_DONE		
0x1E	WRITE_DATASIZE_SEC_SNOOPOUT		
0x1F	TRANS_REQUEST_READ_SNOOPOUT		
0x20	DELAY_SEC_DONE		
0x21	TRANS_REQUEST_READ		

Table 14-52 shows the bit positions of each potential error. Multiple errors are possible.

Table 14-52. Crypto-Channel Pointer Status Register Error Field Definitions

Value	Error
48	DOF - Double Fetch Fifo write overflow Error.This bit is set when the Crypto-channel Fetch Fifo is full, SOF is set, and another write has been made to the fetch FIFO. When this bit is set the crypto-channel will stop, and an error interrupt will be activated. The Channel will not start again until a Continue or Reset is given via the CCCR Register. This bit can be cleared by writing '1' to this CCPSR bit.
49	SOF - Single Fetch Fifo write overflow Error. This bit is set when the Channel Fetch Fifo is full and another write has been made to the Fetch Fifo. The Crypto-channel will set this bit and activate an error interrupt. The Crypto-channel continues processing, but the descriptor pointer is lost. The host must clear this bit by writing '1' to this CCPSR bit.
50	MDTE- A Master Data Transfer Error was received from the master bus interface. When the SEC, while acting as a bus master, detects an error, the controller passes the error to the channel in use. The channel halts and activates an interrupt. The channel can only be restarted by writing a '1' to the Continue or Reset bit in the Crypto-Channel Configuration Register, or resetting the whole SEC.
51	Scatter/Gather Data Length Zero Error- A zero length Scatter/Gather data pointer was detected.
52	Fetch Pointer Zero Error- An all zero Fetch Pointer was detected.
53	Illegal descriptor header- Possible causes of an illegal descriptor header are: • Invalid primary EU indicated by op0 field in descriptor header. • Invalid secondary EU indicated by op1 field in descriptor header. • Descriptor type field in descriptor header indicates secondary EU transaction when not in snoop mode
54	Invalid EU assignment request- Indicates the channel has been assigned one or more EUs not requested by the descriptor header.
55	EU error detected. An EU assigned to this channel has generated an error interrupt. This error may also be reflected in the controller's interrupt status register.
56	Gather boundary error- Indicates a gather pointer straddles both a primary and secondary EU's data transfer.
57	Gather return/length error- Indicates the total data size covered by a gather link table did not match the total data size from the main descriptor.
58	Scatter Boundary Error- Indicates a scatter pointer straddles both a primary and secondary EU's data transfer.
59	Scatter Return/Length Error- Indicates the total data size covered by a scatter link table did not match the total data size from the main descriptor.

NOTE

EU error bit (ERROR[0]) can only be cleared by first clearing the error source in the assigned EU which caused it to be set.

Table 14-53 shows the possible values of the PTR_DW field in the CCPSR.

Table 14-53. Channel Pointer Status Register PTR_DW Field Values

Value	Error
0x00	Processing Header or pointer dword 0
0x01	Processing pointer dword 1
0x02	Processing pointer dword 2
0x03	Processing pointer dword 3
0x04	Processing pointer dword 4

Table 14-53. Channel Pointer Status Register PTR_DW Field Values (continued)

Value	Error
0x05	Processing pointer dword 5
0x06	Processing pointer dword 6
0x07	Complete (or not yet begun) processing of header dword and pointer dwords
0x08– 0xFF	Reserved

14.6.1.3 Crypto-Channel Current Descriptor Pointer Register (CDPR)

The CDPR, shown in Figure 14-63, contains the address of the descriptor which the crypto-channel is currently processing. CPDR, along with the PAIR_PTR in the CCPSR, can be used to determine if a new descriptor can be safely inserted into a chain of descriptors.

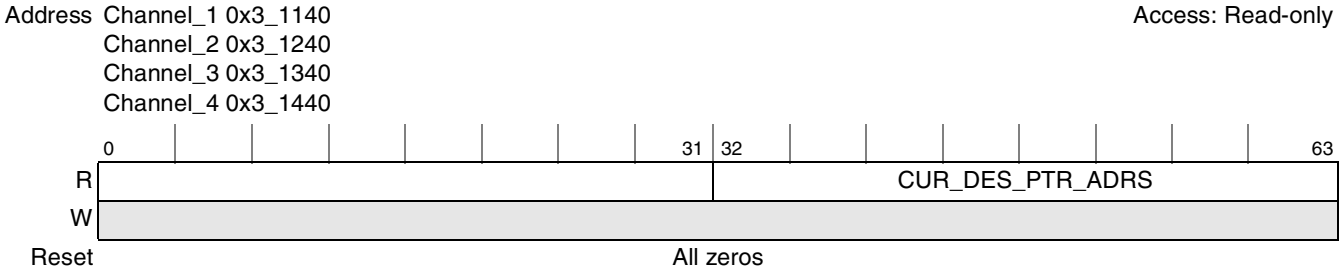


Figure 14-63. Crypto-Channel Current Descriptor Pointer Register

The bits in the current descriptor pointer register perform the functions described in Table 14-54.

Table 14-54. Crypto-Channel Current Descriptor Pointer Register Signals

Bits	Name	Description
0–31	—	Reserved — Set to zero.
32–63	CUR_DES_PTR_ADRS	Current Descriptor Pointer Address. Pointer to system memory location of the current descriptor. This field reflects the starting location in system memory of the descriptor currently loaded into the DB. This value is updated whenever the channel requests a fetch of a descriptor from the controller. The value from the Fetch FIFO is transferred to the current descriptor pointer register immediately after the fetch is completed. This address will be used as the destination of the writeback of the modified header dword, if header writeback notification is enabled.

14.6.1.4 Fetch FIFO (FF)

Each channel contains a fetch FIFO to store a queue of pointers to descriptors which the channel will process.

The fetch FIFO, shown in Figure 14-64, contains the addresses of the first byte of descriptors to be processed. In typical operation, the host CPU will create a descriptor in memory containing all relevant mode and location information for the SEC, then ‘launch’ the SEC by writing the address of the descriptor to the fetch FIFO.

The fetch address is written into the FIFO only if the write includes the least significant byte (bits 56–63).

The fetch FIFO can hold up to 24 descriptor pointers at a time. When the end of the current descriptor is reached, the descriptor pointed to by the next location in the Fetch FIFO will be read to launch the next descriptor. Writing a descriptor pointer to the fetch FIFO while the FIFO is full will result in a single overflow interrupt to advise the user that the descriptor pointer was not successfully written to the fetch FIFO. The channel will continue processing and software can check the fetch FIFO counter in the crypto-channel pointer status register before attempting to re-enqueue the descriptor pointer. If a second descriptor pointer is written to the fetch FIFO before the single the single overflow error is cleared, the channel will generate a double overflow error interrupt and stop processing descriptors. The channel can be restarted by setting the Continue bit in the crypto-channel configuration register, or completely reset by writing the Reset bit in the same register.

Writing a fetch address of zero to the fetch FIFO causes the channel to generate an error and to stop.

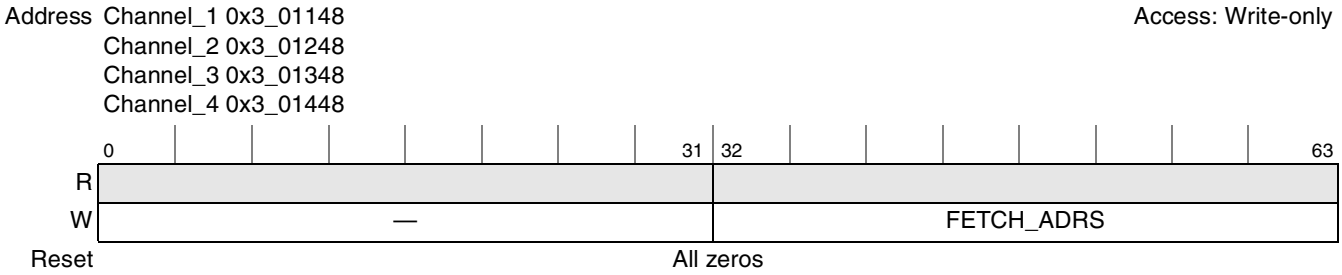


Figure 14-64. Fetch FIFO

Table 14-55 describes the Fetch FIFO fields.

Table 14-55. Fetch FIFO Field Descriptions

Bits	Name	Description
0–31	—	Reserved — Set to zero.
32–63	FETCH ADRS	Fetch Address. Pointer to system memory location of a descriptor the host wants the SEC to fetch.

14.6.1.5 Descriptor Buffer (DB)

The descriptor buffer (DB) consists of 8 dword registers (DB0–DB7), and contains the current descriptor being processed by the channel. These registers are read-only, since the descriptor is always fetched from system memory.

For more information about the fields in a descriptor, see Section 14.4.1, “Descriptor Structure.”

	0 15	16	17 23	24 31	32 63
DB0	Header				Reserved
DB1	Length0	J1	Extent0	—	Pointer0
DB2	Length1	J2	Extent1	—	Pointer1
DB3	Length2	J3	Extent2	—	Pointer2
DB4	Length3	J4	Extent3	—	Pointer3
DB5	Length4	J5	Extent4	—	Pointer4
DB6	Length5	J6	Extent5	—	Pointer5
DB7	Length6	J7	Extent6	—	Pointer6
Address	Channel_1 0x3_1180–0x3_11BF, Channel_2 0x3_1280–0x3_12BF, Channel_3 0x3_1380–0x3_13BF, Channel_4 0x3_1480–0x3_14BF				

Figure 14-65. Data Packet Descriptor Buffer

14.6.2 Interrupts

The crypto-channel can assert both DONE and ERROR interrupts to the controller. When the interrupt generation conditions have been met, the crypto-channel will assert the appropriate interrupt. The status of the registered crypto-channel interrupts are available in the controller interrupt status register. The registered interrupts can cleared by writing to the controller interrupt clear register. The crypto-channel does not have an internal interrupt mask bit and error interrupts are always asserted to the controller. The controller can be programmed to mask channel interrupts to the host via its interrupt mask register (IMR). See Section 14.7.2.1, “Interrupt Mask Register (IMR).”

14.6.2.1 Channel Done Interrupt

Whether and when a Channel DONE Interrupt is generated depends on the setting of the Crypto-Channel Configuration Register NT and CDIE bits in the CCCR (see Figure 14-60). Assuming the CDIE (Channel Done Interrupt Enable) is set, the channel will generate an interrupt after every successfully completed descriptor (Notification Type set to Global), or after each successfully completed descriptor with the DN (Done Notification) bit set in the header word of the descriptor.

The Controller will queue multiple Channel Done interrupts if an interrupt is not cleared before the next Channel Done interrupt is issued. If there are multiple interrupts, the controller will re-assert the Channel Done interrupt one cycle after the previous Channel Done interrupt is cleared.

14.6.2.2 Channel Error Interrupt

The Channel Error Interrupt is generated when an error condition occurs during descriptor processing. The channel error interrupt will be asserted as soon as the error condition is detected. The type of error condition is reflected the ERROR field of the Channel Pointer Status Register (CPSR). Refer to Table 14-52 for a complete listing of error types.

14.6.2.3 Channel Reset

Channel reset is asserted when the host sets the RESET bit in the crypto-channel configuration register (CCCR). The effect of software reset on the channel varies according to what the channel is doing when the bit is set:

- If the RESET bit is set while the crypto-channel is requesting a EU assignment from the controller, the crypto-channel will cancel its request by asserting the release output signals. The crypto-channel will then reset all the registers, clear the RESET bit and return the control state machine to the idle state.
- If the RESET bit is set after the crypto-channel has been dynamically assigned a EU, the channel will request a write from the controller to set the software reset bit of the EU. A write to reset the secondary (MDEU) EU will also be requested if one has been reserved for snooping. The crypto-channel will then assert the appropriate release output signal to notify the controller that the channel has finished with the reserved EU(s). The crypto-channel will then reset all the registers, clear the RESET bit and return the control state machine to the idle state.

14.7 Controller

The controller within the SEC is responsible for overseeing the operations of the execution units (EUs), the interface to the host processor, and the management of the crypto-channels. The controller interfaces to the host via the master/slave bus interface and to the channels and EUs via internal buses. All transfers between the host and the EUs are moderated by the controller. Some of the main functions of the controller are as follows:

- Arbitrate and control accesses to the bus
- Control the internal bus accesses to the EUs
- Arbitrate and assign EUs to the crypto-channels
- Monitor interrupts from channels and pass to host
- Realign read and write data to the proper byte alignment

14.7.1 Controller Registers

The controller registers are described in detail in the following sections.

14.7.2 EU Assignment Status Register (EUASR)

The EUASR, shown in Figure 14-66, is used to check the assignment status of an EU to a particular channel. When an EU is already assigned, it is inaccessible to any other channel.

A four-bit field (see Table 14-56) indicates the channel to which an EU is assigned.

Offset 0x3_1028 Access: Read-only

	0–3	4–7	8–11	12–15	16–19	20–23	24–27	28–31
R	—	AFEU	—	MDEU	—	AESU	—	DEU
W								
Reset	1 1 1 1	0 0 0 0	1 1 1 1	0 0 0 0	1 1 1 1	0 0 0 0	1 1 1 1	0 0 0 0

	32–47	48–51	52–55	56–59	60–63
R	—	—	PKEU	—	RNG
W					
Reset	‘0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1	1 1 1 1	0 0 0 0	1 1 1 1	0 0 0 0

Figure 14-66. EU Assignment Status Register (EUASR)

Table 14-56 shows the EU channel assignments.

Table 14-56. Channel Assignment Value

Value	Channel
0x0	No channel assigned
0x1	Channel 1
0x2	Channel 2
0x3	Channel 3
0x4	Channel 4
0xA–0xE	Undefined
0xF	Unavailable

14.7.2.1 Interrupt Mask Register (IMR)

The SEC controller generates the single interrupt output from all possible interrupt sources. These sources can be masked by the interrupt mask register. If unmasked, the interrupt source value, when active, is captured into the interrupt status register. Figure 14-67 shows the bit positions of each potential interrupt source. Each interrupt source is individually unmasked by setting its corresponding bit. At reset, all bits are masked. The bit fields are described in Table 14-57.

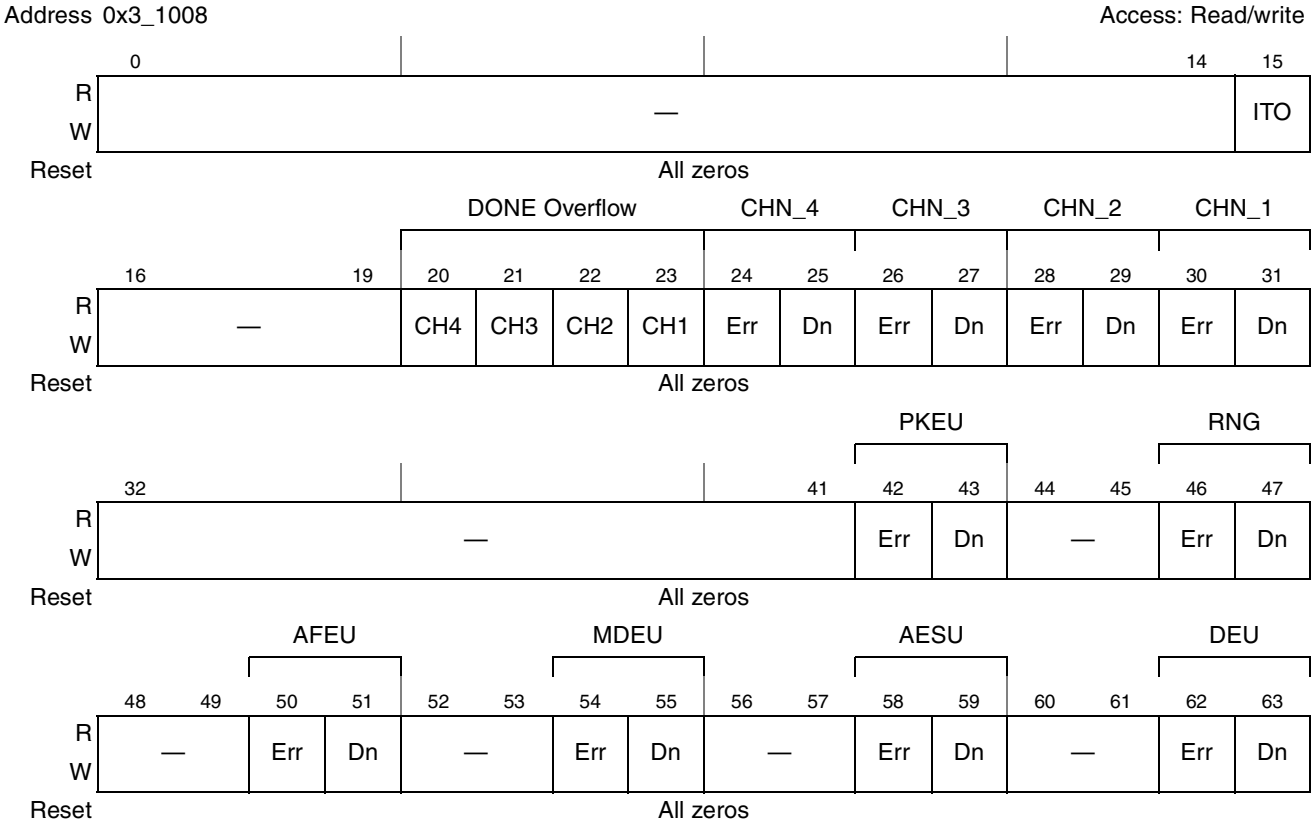


Figure 14-67. Interrupt Mask Register

14.7.2.2 Interrupt Status Register (ISR)

The ISR contains fields representing all possible sources of interrupts. It is cleared either by a reset or by writing the appropriate active ICR bits. Figure 14-68 shows the bit positions of each potential interrupt source. The bit fields are described in Table 14-57.

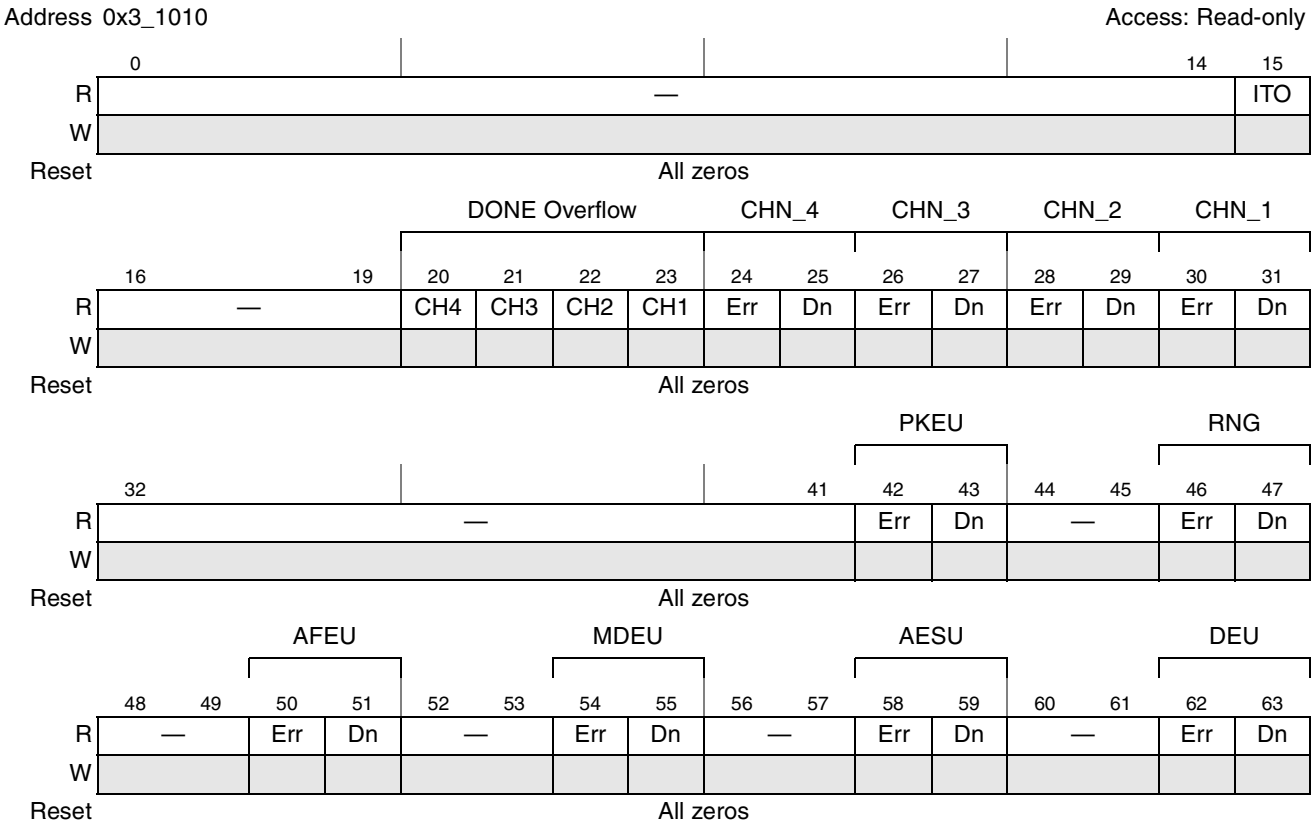


Figure 14-68. Interrupt Status Register

14.7.2.3 Interrupt Clear Register (ICR)

ICR provides a means of clearing the interrupt status register. When a bit in the ICR is written with a 1, the corresponding bit in the ISR is cleared, clearing the interrupt output pin $\overline{\text{IRQ}}$ (assuming the cleared bit in the ISR is the only interrupt source). If the input source to the ISR is a steady-state signal that remains active, the appropriate ISR bit, and subsequently $\overline{\text{IRQ}}$, will be reasserted shortly thereafter. Figure 14-69 shows the bit positions of each interrupt source that can be cleared by ICR. The complete bit definitions for ICR can be found in Figure 14-69. ICR fields are described in Table 14-57.

When an ICR bit is written, it will automatically clear itself one cycle later. That is, it is not necessary to write a ‘0’ to a bit position which has been written with a ‘1.’

NOTE

Interrupts are registered and sent based upon the conditions which cause them. If the cause of an interrupt is not removed, the interrupt will return a few cycles after it has been cleared using the ICR.

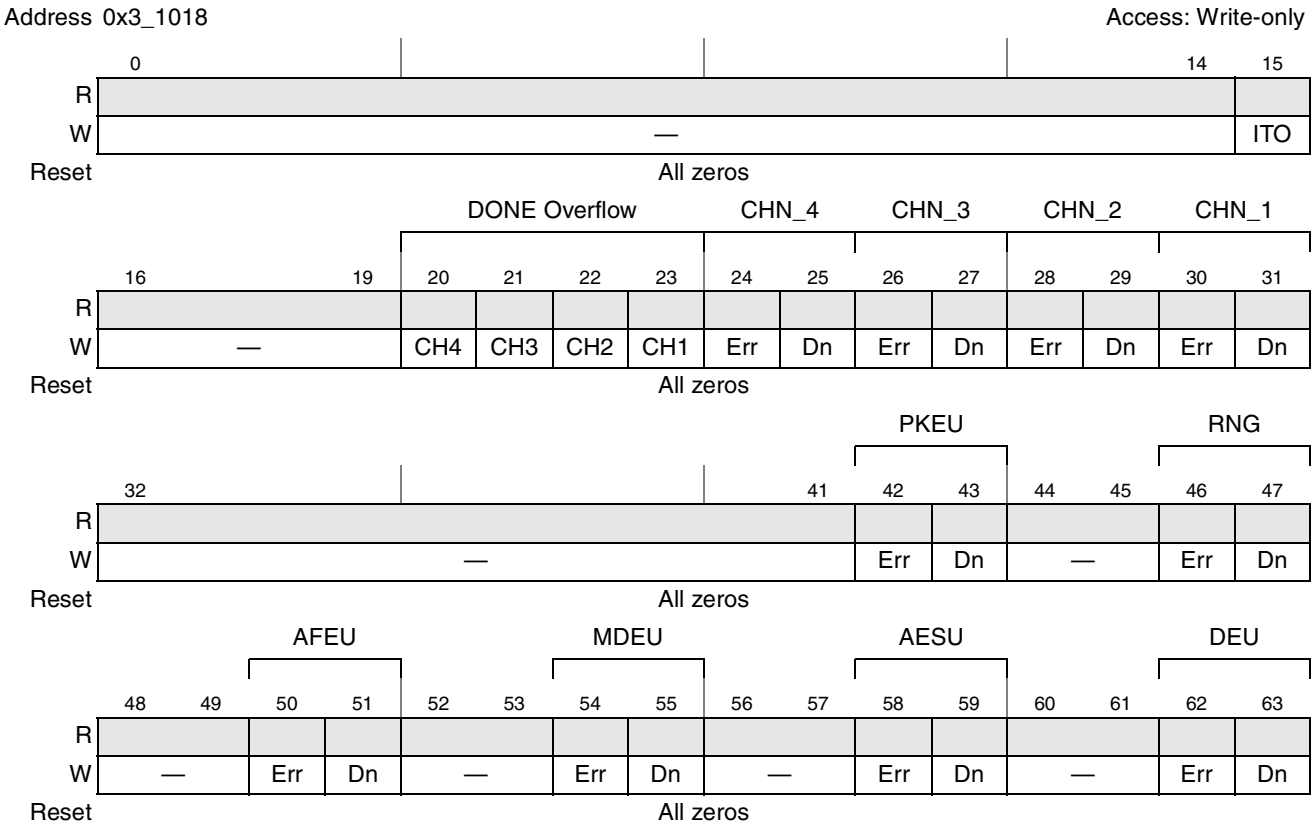


Figure 14-69. Interrupt Clear Register

Table 14-57 describes the interrupt mask, status, and clear register fields.

Table 14-57. Interrupt Mask, Status, and Clear Register Fields

Bits	Name	Description
15	ITO	Internal Time Out 0 No Internal Time Out 1 An Internal Time Out was detected The Internal Time Out interrupt is triggered by the controller if a slave access to an SEC register does not result in successful data transfer within 16 clock cycles. With ITO enabled the SEC controller terminates the transaction and signals and interrupt.
20–23	Done Overflow	Done overflow 0 No Done Overflow 1 Done Overflow Error. Indicates that more than 15 Done interrupts were queued from the interrupting channel without an interrupt clear.
Multiple	CHN_Err_Dn	Each of the 4 channels has Error & Done bits. 0 No error detected. 1 Error detected. Indicates that execution unit status register must be read to determine exact cause of the error. 0 Not DONE. 1 DONE bit indicates that the interrupting channel or EU has completed its operation.

Table 14-57. Interrupt Mask, Status, and Clear Register Fields

Bits	Name	Description
Multiple	EU_Err_Dn	Each of the execution units has Error & Done bits. 0 No error detected. 1 Error detected. Indicates that execution unit status register must be read to determine exact cause of the error. 0 Not DONE. 1 DONE bit indicates that the interrupting channel or EU has completed its operation.
0–14,16–19, 32–41,44–45, 48–49,52–53, 56–57,60–61	—	Reserved, set to zero.

14.7.2.4 ID Register

The read-only ID register, shown in Figure 14-70, contains a 64-bit value that uniquely identifies the version of the SEC 2.4. The ID value is always 0x0000_0000_0000_0041, indicating that this is the first version of the SEC 2.4.

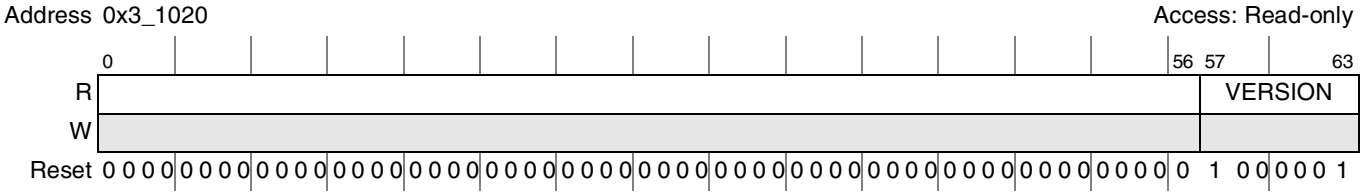

Address 0x3_1020
Access: Read-only
0
56 57
63
R
W
VERSION
Reset 0 0 0

Table 14-58 describes the master control register signals.

Table 14-58. Master Control Register Signals

Bits	Name	Description
0–21	—	Reserved
22–23	Priority	Priority on Master Bus. The setting of these bits determines the transaction priority level the SEC asserts to the device internal arbiter. The SEC does not dynamically alter its priority level based on system congestion or SEC utilization, however software may change the SEC priority level in realtime. 00 - Lowest Priority (default) 01 - Next Lowest Priority 10 - Next Highest Priority 11 - Highest Priority
24–29	—	Reserved
30	GI	Global Inhibit. 0 Master will always drive GBL_B active (low), meaning that the transactions are global and cache snooping is needed in order to enforce coherency. 1 Master will always drive GBL_B inactive (high).
31	SWR	Software Reset. Writing 1 to this bit will cause a global software reset. Upon completion of the reset, this bit will be automatically cleared. 0 Don't reset 1 Global Reset **Note:** Warning: Certain SEC interrupts are not fully cleared by writing this bit. If SEC interrupts are pending, it is recommended that the user set this bit twice (two consecutive writes) to completely reset the SEC.
32–39	CHN3_EU_PR_CNT	Channel 3 EU Priority Counter. This counter is used by the controller to determine when Channel 3 has been denied access to a requested EU long enough to warrant immediate elevation to top priority. **Note:** If set to zero, the CHN4_EU_PR_CTR must also be set to zero, and the controller will assign EUs on a pure round robin basis. If set to non-zero, CHN4_EU_PR_CTR must also be set to a different, non-zero value.
40–47	CHN4_EU_PR_CNT	Channel 4 EU Priority Counter. This counter is used by the controller to determine when Channel 4 has been denied access to a requested EU long enough to warrant immediate elevation to top priority. **Note:** If set to zero, the CHN3_EU_PR_CTR must also be set to zero, and the controller will assign EUs on a pure round robin basis. If set to non-zero, CHN3_EU_PR_CTR must also be set to a different, non-zero value.
48–55	CHN3_BUS_PR_CNT	Channel 3 Bus Priority Counter.This counter is used by the controller to determine when Channel 3 has been denied access to the bus long enough to warrant immediate elevation to top priority. **Note:** If set to zero, the CHN4_BUS_PR_CTR must also be set to zero, and the controller will assign access to the bus on a pure round robin basis. If set to non-zero, CHN4_BUS_PR_CTR must also be set to a different, non-zero value.
56–63	CHN4_BUS_PR_CNT	Channel 4 Bus Priority Counter. This counter is used by the controller to determine when Channel 4 has been denied access to a needed on-chip resource long enough to warrant immediate elevation to top priority. **Note:** If set to zero, the CHN3_BUS_PR_CTR must also be set to zero, and the controller will assign access to the bus on a pure round robin basis. If set to non-zero, CHN3_BUS_PR_CTR must also be set to a different, non-zero value.

14.7.2.6 EU Access

Assignment of a EU function to a channel is done dynamically. With dynamic assignment, the channel requests a EU function, the controller checks to see if the requested EU function is available, and if it is, the controller grants the channel assignment of the EU.

If an EU is available for a channel when requested, the controller will assert the grant signal pertaining to the request from the channel. The grant signal will remain asserted until the channel is done and releases the EU.

14.7.2.7 Multiple EU Assignment

In some cases, a channel may request two EUs. The channel will do this by first requesting the primary EU, then requesting the secondary EU. Once the controller has granted both EUs, this channel is then capable of requesting that the secondary EU snoop the bus. Snooping is described in Section 14.8.3, “Snooping by Caches”.

In all cases, the controller assigns the primary EU to a requesting channel as the EUs become available. The controller does not wait until both EUs are available before issuing any grants to a channel which is requesting two EU functions.

14.7.2.8 Multiple Channels

Since there are multiple channels in the SEC, the controller must arbitrate for access to the execution units. To accomplish this, the controller implements an arbiter for each channel.

Each arbiter acts on either a weighted priority-based or round-robin scheme, depending on the values of CHN3_EU_PR_CNT and CHN4_EU_PR_CNT. If both CHN3_EU_PR_CNT and CHN4_EU_PR_CNT are set to a non-zero value, the arbiter will implement the weighted priority scheme. Otherwise, the arbitration will be round-robin. Setting only one of the CHN_EU_PR_CNT fields to a non-zero value causes unpredictable operation.

14.7.2.9 Priority Arbitration

When arbitrating on the priority scheme, the priority will be as follows:

- Channel 1—Highest priority
- Channel 2—Second highest priority, unless CHN3_EU_PR_CNT or CHN4_EU_PR_CNT expired
- Channel 3—Third priority, unless CHN4_EU_PR_CNT expired
- Channel 4—Lowest priority, until CHN4_EU_PR_CNT expired

For channels 1–4, the priority is channel 1, channel 2, channel 3, and channel 4, in that order. In order to prevent channels 3 and 4 from being locked out, the CHN3_EU_PR_CNT and CHN4_EU_PR_CNT fields are implemented in the Master Control Register. The value of these fields determines how many times channel 3 or channel 4 can be refused access to an EU in favor of a higher priority channel. A counter is implemented in the arbiter for each of these entities. When the channel has lost arbitration the number of times specified in its CHN_EU_PR_CNT field, then that channel has the 2nd highest priority when the requested EU becomes available. CHN1 always has the highest priority, but it cannot make back to back

requests, so the 2nd highest priority channel will be serviced upon completion of the current CHN1 operation.

It is permissible for the CHN_EU_PR_CNT values to be different from the CHN_BUS_PR_CNT values, i.e., EU access may be prioritized, while bus access is pure round robin, and vice-versa.

14.7.2.10 Round-Robin Snapshot Arbiters

The controller implements seven ‘snapshot’ arbiters, one for each EU function, and one for the bus. Each arbiter takes a snapshot of the requests for its function. If there are requests, then the arbiter satisfies those requests via a round-robin scheme as the resource becomes available. When all requests have been satisfied, the arbiter takes another snapshot.

14.8 Bus Interface

The controller in the SEC (refer to Section 14.7, “Controller”) has the ability to be a bus master or a slave. This means that the controller can issue read and write commands to the bus, and it can also be written to and read from by the host.

The controller is the sole bus master in the SEC. All other modules are slave only devices. A channel may request access to system resources including the bus. In these cases, the channel must provide the starting address of the transfer for the bus(es) requested. All subsequent addresses are generated by the controller. All addresses will be sequential.

14.8.1 Bus Access

The controller attempts to maximize bus utilization by grouping outstanding bus requests from the channels by request type. The Controller will push all write requests to the Bus Interface, followed by all read requests, then repeat. Within a request type, the Controller will granted bus access via the same scheme that is used for granting EUs. When the CHN_BUS_PR_CNT values of both channel 3 and 4 are set to zero, round robin operation is in effect. In this case, the snapshot arbiter samples the requests for the bus, then grants those requests as the bus becomes available. For example, if channels 1, 2, and 4 are requesting bus access at a given time, the snapshot arbiter will register the three requests and ignore further requests. The buses will be granted to channel 1 until its transfer is completely satisfied. Then the buses will be granted to channel 2 until channel 2’s transfer is completely satisfied. Finally, the buses will be granted to channel 4 until that transfer is completely satisfied. Then another snapshot of requests will be taken. Refer to Section 14.7.2.9, “Priority Arbitration,” for more information.

14.8.1.1 Master Read

The sequence for master read access is as follows:

1. Channel asserts its bus read request.
2. Channel furnishes address and transfer length.
3. Controller acknowledges request to channel.
4. Controller asserts request to Master interface.
5. Controller waits for bus read to begin.

6. When bus read begins, controller receives data from the master interface and performs a write to the appropriate internal address using the address supplied by the channel. Data may be realigned byte-wise by the controller if either:
   - the read did not begin on a 32-bit word boundary, or
   - the previous write to an execution unit's input FIFO did not end on a 32-bit word boundary
7. Transfer continues until the bus read is completed and the controller has written all data to the appropriate internal address. The master interface will continue making bus requests until the full data length has been read.

14.8.1.2 Master Write

Master writes are performed by transferring data from one of the EUs to the output FIFO in the controller, then transferring the data from the FIFO to the bus when the bus is granted to the controller. The sequence for a master bus write access is as follows:

1. Channel requests the bus from controller.
2. Channel furnishes address, transfer length.
3. Controller acknowledges request to channel.
4. Controller loads the write data into its FIFO, and waits for the bus to become available.
5. When the bus becomes available, controller writes data from its FIFO to the master interface.
6. Transfer continues until the bus write is completed and the controller has read all data from the appropriate internal address. The master interface will continue making bus requests until the full data length has been written.

It is probable that multiple bus bursts will be required to complete any given request. When a channel has been granted access to the bus, no other internal SEC requests to the bus will be acknowledged until that transfer has been fully satisfied; i.e., all bytes have been transferred.

14.8.1.2.1 Slave Access

As a bus slave, the controller simply responds to read and write commands from the bus. When it receives a write command, the controller takes the data from the slave interface and sends it to the internal location indicated by the address. For a read, the controller goes to the internal location and fetches the requested data from the specified address. The SEC internal memory space must be accessed modulo-4 boundaries to avoid invalid data or unpredictable operation.

14.8.2 Bus Arbitration Priority

Transaction priority is configured for all channels in the SEC master control register (MCR[Priority]) as shown in Figure 14-71. The SEC does not dynamically adjust its transaction priority, however system software can dynamically adjust SEC transaction priority, with the change in priority taking effect immediately.

14.8.3 Snooping by Caches

If defined as global, SEC transactions can be snooped by the MPC8349EA cache. This definition is programmed in the master control register MCR[GI]. See 14.7.2.5, “Master Control Register (MCR),” for more details. Note that SEC transactions are defined as global by default.

14.8.4 Interrupts

The SEC generates a single interrupt to the programmable interrupt controller. Interrupts from the SEC are reported to the CPU if the mask bit in SIMSR_H[SEC] is set.

The user can control which events cause an interrupt by configuring the SEC interrupt mask register. These events are as follows:

- Done (of a channel or an execution unit)
- Error (of a channel or an execution unit)

When the user detects an interrupt request from the SEC, it should further read the SEC interrupt status register (ISR) to determine the interrupt source. To clear an interrupt, the user should write 1 to the corresponding bits in the SEC interrupt clear register (ICR).

Events may be further masked per channel by setting or clearing the related fields in the crypto-channel configuration registers. It is suggested that the user leave channel interrupts unmasked, while masking the interrupts from the EUs. Errors or Done signals coming from the EUs eventually cause the channel to signal an ERROR or DONE interrupt. Clearing an interrupt before eliminating the condition that caused the interrupt will cause the interrupt to be asserted again a few cycles later.

14.9 Power–Saving Mode

The SEC can be disabled by clearing SCCR[ENCCM]. See Section 4.5.2.3, “System Clock Control Register (SCCR),” for more information.

Chapter 15
Three-Speed Ethernet Controllers

This chapter describes the two three-speed Ethernet controllers of the MPC8349EA. The two controllers are referenced as TSEC1 and TSEC2.

15.1 Introduction

The Ethernet IEEE 802.3 protocol is widely used on LANs that are based on the carrier-sense multiple access/collision detect (CSMA/CD) approach. Because Ethernet and IEEE 802.3 protocols are similar and can coexist on the same LAN, both are referred to as Ethernet in this manual, unless otherwise noted. 10/100 Ethernet provides increased Ethernet speed from 10 to 100 megabits per second (Mbps) and provides a simple, cost-effective option for backbone and server connectivity. This three-speed Ethernet controller (TSEC) also implements a gigabit Ethernet protocol, which builds on top of the Ethernet protocol, but increases speed tenfold over 10/100 Ethernet to 1000 Mbps, or one gigabit per second (Gbps).

Gigabit Ethernet looks identical to Ethernet from the data link layer upward but it uses the ANSI X3T11 FiberChannel FC-0 (interface and media) and FC-1 (encode/decode) for the PHY Layer. In this manner, the standard takes advantage of the existing high-speed physical interface technology of FiberChannel while maintaining the IEEE 802.3 Ethernet frame format, backward compatibility for installed media, and the use of full- or half-duplex CSMA/CD.

The Ethernet protocol implements the bottom two layers of the open systems interconnection (OSI) 7-layer model, that is, the data link and physical sublayers. Figure 15-1 shows the typical Ethernet protocol stack and the relationship to the OSI model.

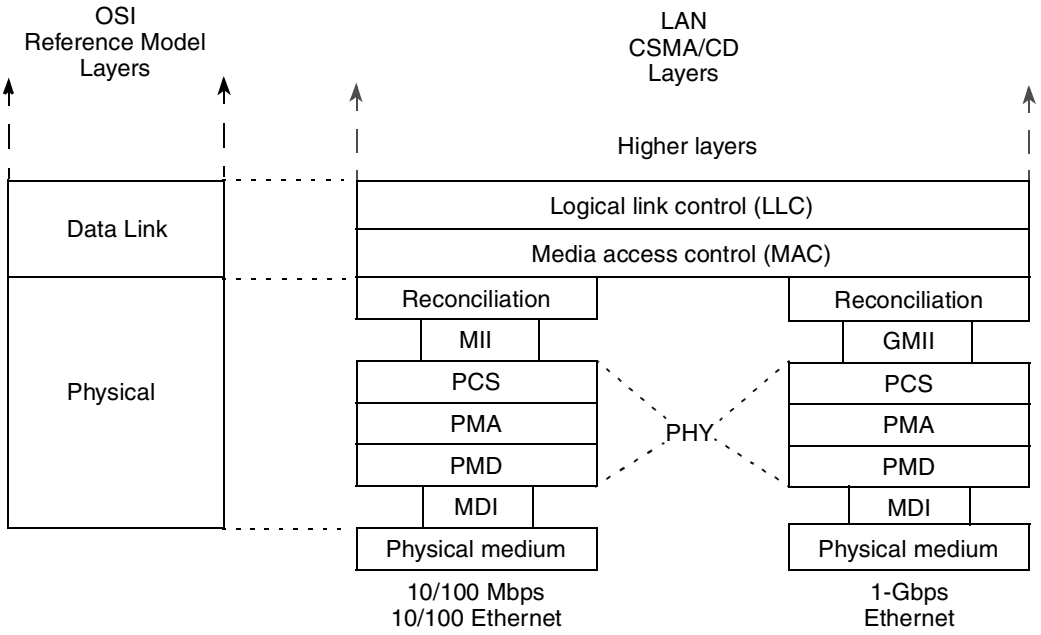


Figure 15-1. Ethernet Protocol in Relation to the OSI Protocol Stack

Gigabit Ethernet provides the following sublayers:

- Media access control (MAC) sublayer—The MAC sublayer provides a logical connection between the MAC and its peer station. Its primary responsibility is to initialize, control, and manage the connection with the peer station.
- Reconciliation sublayer—The reconciliation sublayer acts as a command translator. It maps the terminology and commands used in the MAC layer into electrical formats appropriate for the physical layer entities.
- MII (media-independent interface) sublayer—The MII sublayer provides a standard interface between the MAC layer and the physical layer for 10/100 Mbps operations. It isolates the MAC layer and the physical layer, enabling the MAC layer to be used with various implementations of the physical layer.
- GMII (gigabit media-independent interface) sublayer—The GMII sublayer provides a standard interface between the MAC layer and the physical layer for 1-Gbps operation. It isolates the MAC layer and the physical layer, enabling the MAC layer to be used with various implementations of the physical layer.
- PCS (physical coding sublayer)—The PCS sublayer is responsible for encoding and decoding data stream to and from the MAC sublayer. Medium (1000BASEX) 8B/10B coding is used for fiber. Medium (1000BASET) 8B1Q coding is used for unshielded twisted pair (UTP).
- PMA (physical medium attachment) sublayer—The PMA sublayer is responsible for serializing code groups into a bit stream suitable for serial bit-oriented physical devices (SerDes) and vice versa. Synchronization is also performed for proper data decoding in this sublayer. The PMA sits between the PCS and the PMD sublayers. For fiber medium (1000BASEX) the interface on the PMD side of the PMA is a 1-bit 1250-MHz signal, while on the PMA's PCS side the interface is a 10-bit interface (TBI) at 125 MHz. The TBI is an alternative to the GMII interface. If the TBI is

used, the gigabit Ethernet controller must be capable of performing the PCS function. For UTP medium, the PMD interface side of the PMA consists of four pair of 62.5-MHz PAM5 encoded signals, while the PCS side provides the 1250 Mbps input to a 8B1Q4 PCS.

- PMD (physical medium dependent) sublayer—The PMD sublayer is responsible for signal transmission. The typical PMD functionality includes amplifier, modulation, and wave shaping. Different PMD devices may support different media.
- MDI (medium-dependent interface) sublayer—MDI is a connector. It defines different connector types for different physical media and PMD devices.

Figure 15-2 describes the different physical interface standards.

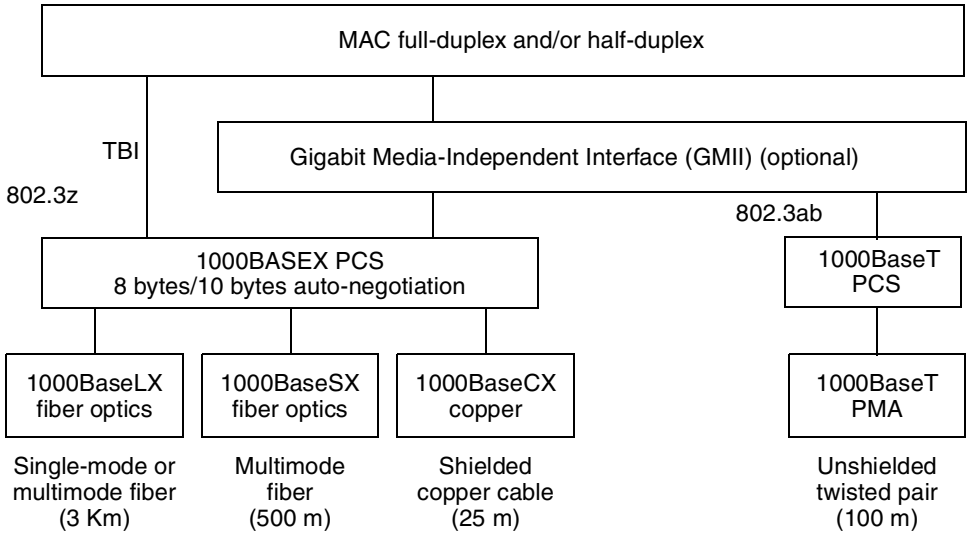


Figure 15-2. 802.3z and 802.3ab Physical Standards

Ethernet/IEEE 802.3 frames are based on the frame structure shown in Figure 15-3. The term ‘packet’ is sometimes used to refer to the frame plus the preamble and start frame delimiter (SFD).

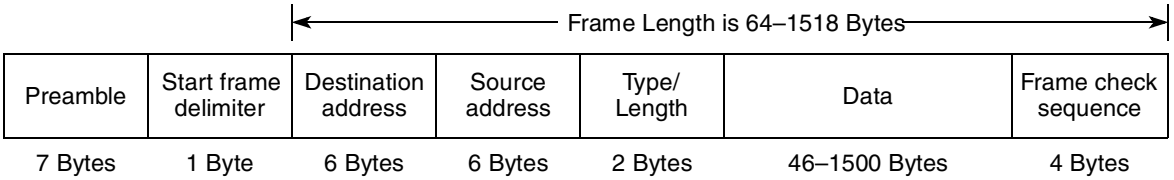


Figure 15-3. Ethernet/IEEE 802.3 Frame Structure

The elements of an Ethernet frame are as follows:

- Preamble—The 7-byte preamble of alternating ones and zeros used for receiver timing synchronization (each byte containing the value 0x55).
- Start frame delimiter (SFD)—A sequence of 0xD5 (10101011 because the bit ordering is lsb first) indicates the beginning of the frame.
- 48-bit destination address (DA)—The first bit identifies the address as an individual address (0) or a group address (1). The second bit is used to indicate whether the address is locally-defined (1) or globally-defined (0).

- 48-bit source address (SA)—(Original versions of the IEEE 802.3 specification allowed 16-bit addressing, which has never been used widely.)
- Ethernet type field/IEEE 802.3 length field—The type field signifies the protocol (for example, TCP/IP) used in the rest of the frame. The length field specifies the length of the data portion of the frame. For both Ethernet and IEEE 802.3 frames to exist on the same LAN, the length field must be unique from any type fields used in Ethernet. This limitation requires that a type field be identified by a decimal number equal to or greater than 1536 (0x0600) but less than 65535 (0xFFFF). If the number, however, is between 0 and 1,500 (0x0000 through 0x05DC) then this field indicates the length of the MAC client data. The range from 1,501 to 1,535 (0x5DD through 0x5FF) was intentionally left undefined.
- Data and padding—Padding is optional. It is only needed if the data is smaller than 46 octets (one octet = one byte) to ensure the minimum frame size of 64 octets as specified in the IEEE 802.3 standard. In 802.3x the first two octets of the data field are used as opcode (OP) (pause = 0x0001) and the second two octets are used to transmit a pause time (PT) parameter (pausetime = 0x0000 for on and 0xFFFF for off). In addition, a third two-octet field can be used for an extended pause control parameter (PTE). Because the use of these fields varies with the protocol used, the ability to examine them and report their content can significantly accelerate Ethernet frame processing.
- Frame-check sequence (FCS)—Specifies the standard 32-bit cyclic redundancy check (CRC) obtained using the standard CCITT-CRC polynomial on all fields except the preamble, SFD and CRC.

Figure 15-4 provides additional details of the Ethernet/IEEE 802.3 frame structure.

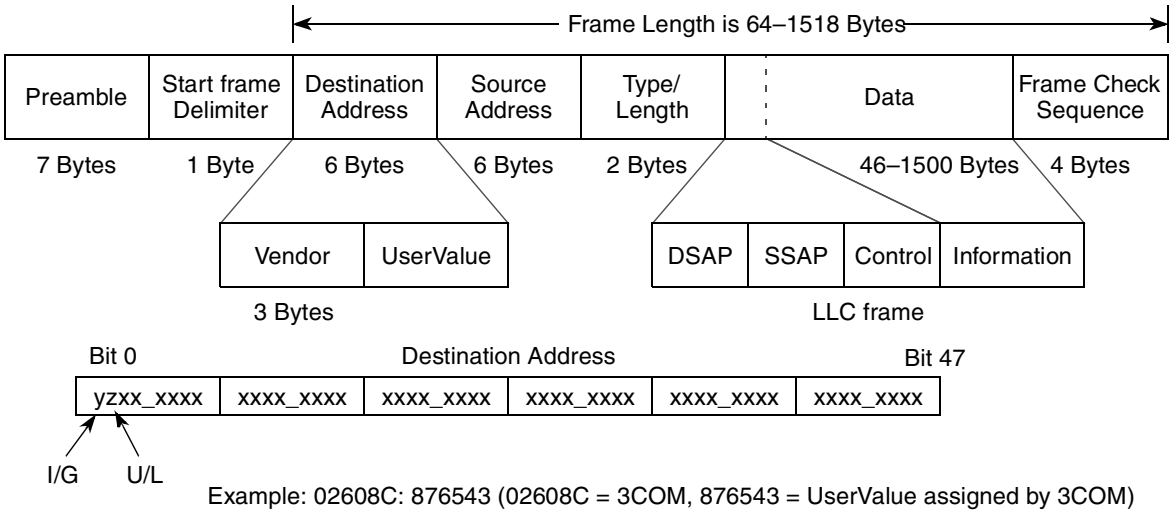


Figure 15-4. Ethernet/IEEE 802.3 Frame Structure with More Details

Relative to Figure 15-4, the IEEE 802.3 section 3.11 (MAC frame format) defines the frame format such that the octets of a frame are transmitted from left to right (preamble first, the FCS last), and the bits of each octet are transmitted least-significant bit (lsb) first. The destination address example shown in Figure 15-4 (02608C:876543) which would normally be written as:

0000 0010 0110 0000 1000 1100 1000 0111 0110 0101 0100 0011

is transmitted bit by bit as:

0100 0000 0000 0110 0011 0001 1110 0001 1010 0110 1100 0010

which is an individual address (because the lsb is cleared) but locally-defined (because the second least-significant bit is set).

When first originated, a type field was used for protocol identification. The IEEE 802.3 specification eliminated the type field, replacing it with the length field. The length field is used to identify the length, in bytes, of the data field. The protocol type in 802.3 frames are held within the data portion of the packet. The logical link control (LLC) is responsible for providing services to the network layer regardless of media type, such as FDDI, Ethernet, token ring, and others. The LLC layer makes use of LLC protocol data units (PDUs) in order to communicate between the media access control (MAC) layer and the upper layers of the protocol stack. Three variables determine access into the upper layers via the LLC-PDU.

The variables include the destination service access point (DSAP), the source service access point (SSAP), and a control variable. The DSAP address specifies a unique identifier within the station providing protocol information for the upper layer. The SSAP provides the same information for the source address.

The LLC defines service access for protocols that conform to the open system interconnection (OSI) model for network protocols. However, many protocols do not obey the rules for those layers and additional information must be added to the LLC in order to provide information regarding those protocols. Protocols that fall into this category include IP and IPX. The method used to provide this additional protocol information is called a subnetwork access protocol (SNAP) frame. A SNAP encapsulation is indicated by the DSAP and SSAP addresses being set to 0xAA and the LLC control field being set to 0x03. If that address is seen, a SNAP header follows. The SNAP header is five bytes long. The first three bytes consist of the organization code (SNAP OUI), which is assigned by the IEEE. The last two bytes become the type value set from the original Ethernet specifications if SNAP OUI = 0, or they become a SNAP protocol identifier if SNAP OUI is non zero.

The three-speed Ethernet controller (TSEC) allows the flexibility to accelerate the identification and retrieval of all the standard and non-standard protocols mentioned above. Figure 15-5 shows the block diagram of the TSEC.

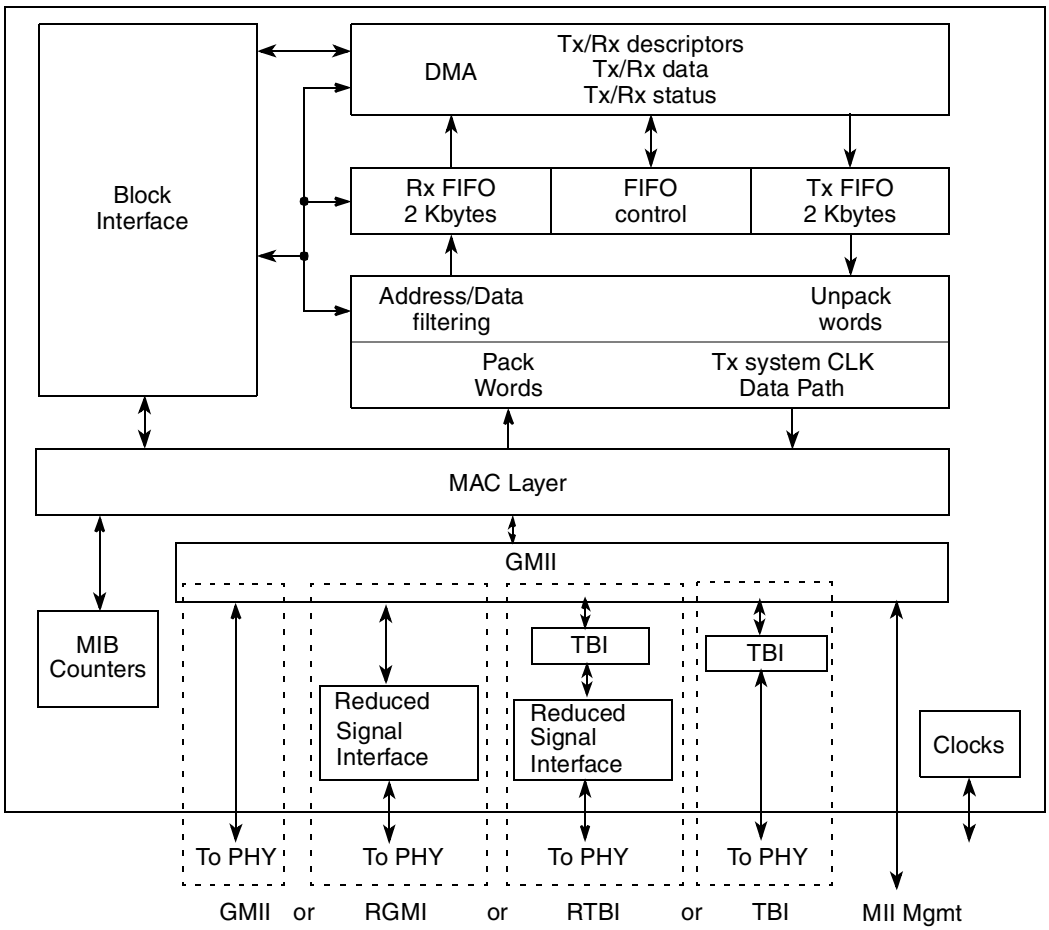


Figure 15-5. TSEC Block Diagram

15.1.1 Three-Speed Ethernet Controller Overview

The TSEC is designed to support 10, 100, and 1000 Mbps Ethernet/802.3 networks and contains the following components:

- Ethernet media access controller (MAC)
- First-in first-out (FIFO) controller
- Direct memory access (DMA) controller
- Ten-bit interface (TBI)
- Register-based statistical module that supports management information base (MIB) remote monitoring (RMON)

The most-significant byte of data in a receive or transmit data buffer corresponds to the most-significant byte of a frame, respectively.

The complete TSEC is designed for single MAC applications. The TSEC supports several standard MAC-PHY interfaces to connect to an external Ethernet transceiver:

- MII interface running at 10/100 Mbps
- GMII interface running at 1 Gbps
- TBI interface that can be connected to a SerDes device for fibre channel applications
- Reduced signal count versions of the GMII (RGMII) and ten-bit (RTBI) interfaces

While most of this document refers to the non-reduced signal count interfaces, it must be understood that these references also apply to the reduced signal count interfaces.

The TSEC software programming model is similar to the MPC8260 (PowerQUICC II) device. Hence, it enables Freescale customers to leverage already implemented Ethernet drivers, reducing the software development cycle.

15.2 Features

The MPC8349EA TSEC includes these distinctive features:

- IEEE 802.3, 802.3u, 802.3x, 802.3z, 802.3ac, 802.3ab compliant
- Support for different Ethernet physical interfaces:
  - 10/100/1Gb IEEE 802.3 GMII
  - 10/100 Mbps IEEE 802.3 MII
  - 10-Mbps IEEE 802.3 MII
  - 1-Gbps IEEE 802.3z TBI
  - 10/100 Mbps RGMII
  - 1-Gbps full-duplex RGMII
  - 1-Gbps RTBI
- Full- and half-duplex support (1 Gbps supports only full-duplex)
- IEEE 802.3 full-duplex flow control (automatic PAUSE frame generation or software programmed PAUSE frame generation and recognition)
- Support for out-of-sequence transmit queue (for initiating flow-control)
- Programmable maximum frame length supports jumbo frames (up to 9.6 Kbytes) and IEEE 802.1 virtual local area network (VLAN) frames
- Retransmission from transmit FIFO following a collision
- Support for CRC generation and verification of inbound/outbound packets
- Address recognition
  - Each exact match can be programmed to be accepted or rejected
  - Broadcast address (accept/reject)
  - Exact match 48-bit individual (unicast) address
  - Hash (256-bit hash) check of individual (unicast) addresses

  - Hash (256-bit hash) check of group (multicast) addresses
  - Promiscuous mode
- Interrupt coalescing subject to a threshold frame counter and/or a threshold timer (Independent functionality for received and transmitted frames)
- RMON statistics support

15.3 Modes of Operation

The primary TSEC operational modes are the following:

- Full- and half-duplex operation

  This is determined by MACCFG2 register's full-duplex bit. 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 (only 10- and 100-Mbps operation; MACCFG2 register's full-duplex bit is cleared), the MAC complies with the IEEE CSMA/CD access method.

  If configured in full-duplex mode (10/100/1Gb operation; MACCFG2 register's full-duplex bit 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 configuring bits MACCFG2[22:23] = 01. The MII is the media-independent interface defined by the 802.3 standard for 10/100 Mbps operation. The speed of operation is determined by the TSEC*n*_TX_CLK and TSEC*n*_RX_CLK signals, which are driven by the transceiver. The transceiver either auto-negotiates the speed or it may be controlled by software via the serial management interface (EC_MDC/EC_MDIO signals) to the transceiver.
- 1-Gbps GMII and TBI interface operation

  The MAC-PHY interface operates in GMII mode by configuring bits MACCFG2[22:23] = 10. The GMII is the gigabit media-independent interface defined by the 802.3 standard for 1-Gbps operation.

  Independently, the MAC-PHY interface can also operate in TBI mode. Note that either the TBI or GMII interface is chosen; not both at the same time. TBI is the ten-bit interface which contains PCS functions (ten-bit encoding/decoding) as defined by the 802.3 standard.

  In reduced signal count mode (RGMII or RTBI), the MAC remains configured in GMII or TBI but the TSEC multiplexes and decodes the input signals and provides the MAC with the expected interface.

  TSEC provides the TSEC*n*_GTX_CLK to the PHY in either GMII or TBI mode of operation.
- Address recognition options

  The options supported are promiscuous, broadcast, exact individual address hash or exact match, and multicast hash match. For detailed descriptions refer to Section 15.5.3.8.1, "Individual Address Registers 0–7 (IADDRn)," Section 15.5.3.8.2, "Group Address Registers 0–7 (GADDRn)," Section 15.5.3.4.1, "Receive Control Register (RCTRL)," and Section 15.6.2.6, "Frame Recognition."

- RMON support

  See Section 15.6.2.5, “RMON Support.”
- Internal loop back

  Internal loop back mode is selected via the loop back bit in the MACCFG1 register. See Section 15.6.2.10, “Internal and External Loop Back,” for details.

15.4 External Signals Description

This section defines the TSEC signals. The buses are described using the bus convention used in IEEE 802.3 because the PHY follows this same convention (that is, TxD[7:0] means 0 is the lsb). Notice that except for external physical interfaces the buses and registers follow a big-endian format.

The TSEC network interface supports multiple options:

- The MII option requires 18 I/O signals (including the EC_MDIO and EC_MDC MII management interface) and supports both a data and a management interface to the PHY (transceiver) device. The MII option supports both 10- and 100-Mbps Ethernet rates.
- The GMII option is a superset of the MII signals and supports a 1-Gbps Ethernet rate.
- The TBI interface shares signals with the GMII interface signals.
- Finally, RGMII and RTBI options are reduced-signal implementations of the GMII and TBI interfaces.

15.4.1 Detailed Signal Descriptions

Table 15-1 contains detailed descriptions of the TSEC interface signals. For RGMII mode details please refer to the Hewlett-Packard reduced gigabit media-independent interface (RGMII) specification version 1.2a, dated 9/22/2000. All other modes follow the IEEE 802.3 standard, 2000 Edition. Input signals not used are internally disabled. Output signals not used are driven low.

Table 15-1. TSEC Signals—Detailed Signal Descriptions

Signal	I/O	Description	
TSEC*n*_COL	I	Collision input. The behavior of this signal is not specified while in full-duplex mode.	
		State Meaning	Asserted/Negated—In MII mode, this signal is asserted upon detection of a collision, and must remain asserted while the collision persists. This signal is not used in the TSEC GMII, TBI, RTBI and RGMII modes.
		Timing	Asserted/Negated—This signal is not required to transition synchronously with TSEC*n*_TX_CLK or TSEC*n*_RX_CLK.
TSEC*n*_CRS	I	Carrier sense input. In TBI mode this signal can be used as SDET (signal detect), an optional signal that some PHYs generate in TBI mode. This signal is not used in the TSEC GMII, RGMII or RTBI modes.	
		State Meaning	Asserted/Negated—In MII mode, TSEC*n*_TX_CLK is asserted while the transmit or receive medium is not idle. In the event of a collision, TSEC*n*_CRS must remain asserted for the duration of the collision.
		Timing	Asserted/Negated—This signal is not required to transition synchronously with TSEC*n*_TX_CLK or TSEC*n*_RX_CLK.

Table 15-1. TSEC Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description
TSEC*n*_GTX_CLK	O	Gigabit transmit clock. This signal is an output from the TSEC into the PHY. In GMII, TBI, or RTBI mode TSEC*n*_GTX_CLK is a 125-MHz clock that provides a timing reference for TX_EN, TXD, and TX_ER. In RGMII mode TSEC*n*_GTX_CLK becomes the transmit clock and provides timing reference during 1000BaseT (125-MHz), 100BaseT (25-MHz) and 10BaseT (2.5-MHz) transmissions. This signal is not used in MII mode.
EC_GTX_CLK125	I	Gigabit transmit 125-MHz source. This signal must be generated externally with a crystal or oscillator, or is sometimes provided by the PHY. In GMII, RGMII, RTBI or TBI mode, EC_GTX_CLK125 is a 125-MHz input into the TSEC and is used to generate all 125-MHz related signals and clocks. This input is not used in MII mode.
EC_MDC	O	Management data clock. In GMII, MII, RGMII, RTBI or TBI mode 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). This clock is generated by dividing the coherent system bus (CSB) clock by eight. The ratio can be modified further by writing to MIIMCFG[29:31]. Note that this signal is used during reset to configure the TSEC interface to reduced-signal or non-reduced-signal mode. See Section 4.3.3, “Loading the Reset Configuration Words,” for more information.
EC_MDIO	I/O	Management data input/output, BIDI **State Meaning**: Asserted/Negated—In GMII, MII, RGMII, RTBI or TBI mode EC_MDIO is a bi-directional signal to input PHY-supplied status during management read cycles and output control during MII management write cycles. **Timing**: Asserted/Negated—This signal is required to be synchronous with the EC_MDC signal.
TSEC*n*_RX_CLK	I	Receive clock. In GMII, MII or RGMII mode, the receive clock TSEC*n*_RX_CLK is a continuous clock (2.5, 25, or 125 MHz) that provides a timing reference for TSEC*n*_RX_DV, TSEC*n*_RXD, and TSEC*n*_RX_ER. In TBI mode, TSEC*n*_RX_CLK is the input for a 62.5-MHz PMA receive clock, 0 split phase with PMA_RX_CLK1 and is supplied by the SerDes. In RTBI mode it is a 125-MHz receive clock.
TSEC*n*_RX_DV	I	Receive data valid. In GMII or MII mode, if TSEC*n*_RX_DV is asserted, the PHY is indicating that valid data is present on the GMII and MII interfaces. In RGMII mode, TSEC*n*_RX_DV becomes RX_CTL. The RX_DV and RX_ERR are received on this signal on the rising and falling edges of TSEC*n*_RX_CLK. In TBI mode, TSEC*n*_RX_DV represents receive code group (RCG) bit 8. Together, with RCG[9] and RCG[7:0], they represents the 10-bit encoded symbol of GMII receive signals. In RTBI mode, TSEC*n*_RX_DV represents receive code group (RCG) bit 4 and 9. On the positive edge of the TSEC*n*_RX_CLK, RCG[4] and RCG[3:0] represents the first half of the 10-bit encoded symbol. On the negative edge of the TSEC*n*_RX_CLK, RCG[9] and RCG[8:4] represents the second half of the 10-bit encoded symbol.

Table 15-1. TSEC Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description
TSEC*n*_ RXD[7:0]	I	Receive data in. In GMII mode, TSEC*n*_RXD[7:4] with TSEC*n*_RXD[3:0], represent one complete octet of data to be transferred from the PHY to the MAC when TSEC*n*_RX_DV is asserted. In TBI mode, TSEC*n*_RXD[7:4] represents RCG[7:4]. Together, with RCG[9:8] and RCG[3:0], they represent the 10-bit encoded symbol of GMII receive signals. In GMII mode, TSEC*n*_RXD[3:0] represents a nibble of data to be transferred from the PHY to the MAC when TSEC*n*_RX_DV is asserted. A completely-formed SFD must be passed across the MII. While TSEC*n*_RX_DV is not asserted, TSEC*n*_RXD has no meaning. In RGMII or RTBI mode, TSEC*n*_RXD[3:0] are received on the rising edge of TSEC*n*_RX_CLK and TSEC*n*_RXD[7:4] are received on the falling edge of TSEC*n*_RX_CLK. In TBI mode, TSEC*n*_RXD[3:0] represents RCG[3:0]. Together, with RCG[9:4], they represent the 10-bit encoded symbol of GMII receive signals.
TSEC*n*_RX_ ER	I	Receive error **State Meaning**: Asserted/Negated—In GMII or MII mode, if TSEC*n*_RX_ER and TSEC*n*_RX_DV are asserted, the PHY has detected an error in the current frame. In TBI mode, TSEC*n*_RX_ER represents RCG[9]. Together, with RCG[8:0], they represent the 10-bit encoded symbol of GMII receive signals. This signal is not used in the TSEC RTBI or RGMII modes.
TSEC*n*_TX_ CLK	I	Transmit clock in. In MII mode, TSEC*n*_TX_CLK is a continuous clock (2.5 or 25 MHz) that provides a timing reference for the TSEC*n*_TX_EN, TSEC*n*_TXD, and TSEC*n*_TX_ER signals. In GMII mode, TSEC*n*_TX_CLK provides the 2.5 or 25-MHz timing reference during 10BaseT and 100Base-T and comes from the PHY. In 1000Base-T this clock is not used and TSEC*n*_GTX_CLK (125 MHz) becomes the timing reference. The TSEC*n*_GTX_CLK is generated in the TSEC and provided to the PHY and the MAC. The TSEC*n*_TX_CLK is generated in the PHY and provided to the MAC. In TBI mode, this signal is PMA receive clock 1, 62.5 MHz, split phase with PMA_RX_CLK0, and is supplied by the SerDes. This signal is not used in the TSEC RTBI or RGMII modes.
TSEC*n*_ TXD[7:0]	O	Transmit data out. In GMII mode, TSEC*n*_TXD[7:4], together with TSEC*n*_TXD[3:0], represent one complete octet of data to be sent from the MAC to the PHY when TSEC_TX_DV is asserted and has no meaning while TSEC*n*_TX_EN is negated. In TBI mode, TSEC*n*_TXD[7:4] represents transmit code group (TCG) bits 7:4. Together, with TCG[9:8] and TCG[3:0], they represent the 10-bit encoded symbol. In GMII or MII mode, TSEC*n*_TXD[3:0] represent a nibble of data to be sent from the MAC to the PHY when TSEC*n*_TX_EN is asserted and have no meaning while TSEC*n*_TX_EN is negated. In RGMII or RTBI mode, TSEC*n*_TXD[3:0] are transmitted on the rising edge of TSEC*n*_TX_CLK, and TSEC*n*_TXD[7:4] are transmitted on the falling edge of TSEC*n*_TX_CLK. In TBI mode, TSEC*n*_TXD[3:0] represents TCG[3:0]. Together, with TCG[9:4], they represent the 10-bit encoded symbol.

Table 15-1. TSEC Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description
TSEC*n*_TX_EN	O	Transmit data valid. In GMII or MII mode, if TSEC*n*_TX_EN is asserted, the MAC is indicating that valid data is present on the GMII's or the MII's TSEC*n*_TXD signals. In RGMII mode, TSEC*n*_TX_EN becomes TX_CTL. TX_EN and TX_ERR are asserted on this signal on rising and falling edges of the TSEC*n*_TX_CLK, respectively. In TBI mode, TSEC*n*_TX_EN represents TCG[8]. Together, with TCG[9] and TCG[7:0], they represent the 10-bit encoded symbol. In RTBI mode, TSEC*n*_TX_EN represents TCG[4]. Together with TCG[9], TCG[3:0] and TCG[8:5], they represent the 10-bit encoded symbol.
TSEC*n*_TX_ER	O	Transmit error. In GMII or MII mode, assertion of TSEC*n*_TX_ER for one or more clock cycles while TSEC*n*_TX_EN is asserted causes the PHY to transmit one or more illegal symbols. Asserting TSEC*n*_TX_ER has no effect while operating at 10 Mbps or while TSEC*n*_TX_EN is negated. This signal transitions synchronously with respect to TSEC*n*_TX_CLK. In TBI mode, TSEC*n*_TX_ER represents TCG[9]. Together, with TCG[8:0], they represents the 10-bit encoded symbol. This signal is not used in the TSEC RTBI or RGMII modes.

15.5 Memory Map/Register Definition

The TSEC uses a software model similar to that employed by the fast Ethernet function supported on the MPC8260 CPM FCC and in the FEC of the MPC860T.

The TSEC device is programmed by a combination of control/status registers (CSR) and buffer descriptors. The CSRs are used for mode control, interrupts, and to access 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 with 32-bit accesses. There is no support for accesses of sizes other than 32 bits.

This section of the document defines the memory map and describes the registers in detail. The buffer descriptor is described in Section 15.6.3, "Buffer Descriptors."

The ten-bit interface (TBI) module MII registers are also described in this section. The TBI registers are defined like PHY registers and, as such, are accessed via the MII management interface in the same way as the PHYs are accessed. For detailed descriptions of the TBI registers (the MII register set for the ten-bit interface) please refer to Section 15.5.4, "Ten-Bit Interface (TBI)."

15.5.1 Top-Level Module Memory Map

The TSEC implementation requires 4 Kbytes of memory-mapped space, of which more than 1 Kbyte is reserved for future expansion. The space is divided into the following sections:

- General control/status registers
- Transmit-specific control/status registers
- Receive-specific control/status registers
- MAC registers

- Event/statistic counters held in the MIB block
- Hash function registers

Table 15-2 defines the top-level memory map.

Table 15-2. Module Memory Map Summary

Address	Function
000–0FF	TSEC general control/status registers
100–2FF	TSEC transmit control/status registers
300–4FF	TSEC receive control/status registers
500–5FF	TSEC MAC registers
600–7FF	TSEC RMON MIB registers
800–8FF	TSEC HASH function registers
900–AFF	Reserved
B00–BFF	TSEC attribute registers
C00–FFF	Future expansion space

15.5.2 Detailed Memory Map—Control/Status Registers

Table 15-3 lists the address, name, and a cross-reference to the complete description of each register. The offsets to the memory map table are defined for both TSECs. That is, TSEC1 starts at 0x2_4000 address offset and TSEC2 starts at 0x2_5000 address offset. The registers for TSEC1 are listed in Table 15-3, but the registers for TSEC2 are not. Note in Table 15-3 that the registers are the same for TSEC2 except that the offset changes from 0x2_4*nnn* to 0x2_5*nnn*. Undefined 4-byte address spaces within offset 0x000–0xFFF are reserved.

Table 15-3. Module Memory Map

Offset	Name	Access[1]	Reset	Section/Page
TSEC1 General Control and Status Registers				
0x2_4000–0x2_400C	Reserved	—	—	—
0x2_4010	IEVENT—Interrupt event register	R/W	0x0000_0000	15.5.3.1.1/15-19
0x2_4014	IMASK—Interrupt mask register	R/W	0x0000_0000	15.5.3.1.2/15-22
0x2_4018	EDIS—Error disabled register	R/W	0x0000_0000	15.5.3.1.3/15-24
0x2_401C	Reserved	R	0x0000_0000	—
0x2_4020	ECNTRL—Ethernet control register	R/W	0x0000_0000	15.5.3.1.4/15-25
0x2_4024	MINFLR—Minimum frame length register	R/W	0x0000_0040	15.5.3.1.5/15-26
0x2_4028	PTV—Pause time value register	R/W	0x0000_0000	15.5.3.1.6/15-27
0x2_402C	DMACTRL—DMA control register	R/W	0x0000_0000	15.5.3.1.7/15-28

Table 15-3. Module Memory Map (continued)

Offset	Name	Access[1]	Reset	Section/Page
0x2_4030	TBIPA—TBI PHY address register	R/W	0x0000_0000	15.5.3.1.8/15-29
0x2_4034–0x2_4088	Reserved	—	—	—
TSEC1 FIFO Control and Status Registers				
0x2_408C	FIFO_TX_THR—FIFO transmit threshold register	R/W	0x0000_0100	15.5.3.2.1/15-30
0x2_4090	Reserved	—	—	—
0x2_4094	FIFO_TX_SP—FIFO transmit space available register	R/W	0x0000_0010	15.5.3.2.2/15-31
0x2_4098	FIFO_TX_STARVE—FIFO transmit starve register	R/W	0x0000_0080	15.5.3.2.3/15-31
0x2_409C	FIFO_TX_STARVE_SHUTOFF—FIFO transmit starve shutoff register	R/W	0x0000_0100	15.5.3.2.4/15-32
0x2_40A0–0x2_40FC	Reserved	—	—	—
TSEC1 Transmit Control and Status Registers				
0x2_4100	TCTRL—Transmit control register	R/W	0x0000_0000	15.5.3.3.1/15-33
0x2_4104	TSTAT—Transmit status register	R/W	0x0000_0000	15.5.3.3.2/15-34
0x2_4108	Reserved	R	0x0000_0000	—
0x2_410C	TBDLEN—TxBD data length register	R	0x0000_0000	15.5.3.3.3/15-34
0x2_4110	TXIC—Transmit interrupt coalescing configuration register	R/W	0x0000_0000	15.5.3.3.4/15-35
0x2_4114–0x2_4120	Reserved	—	—	—
0x2_4124	CTBPTR—Current TxBD pointer register	R	0x0000_0000	15.5.3.3.5/15-36
0x2_4128–0x2_4180	Reserved	—	—	—
0x2_4184	TBPTR—TxBD pointer register	R/W	0x0000_0000	15.5.3.3.6/15-36
0x2_4188–0x2_4200	Reserved	—	—	—
0x2_4204	TBASE—TxBD base address register	R/W	0x0000_0000	15.5.3.3.7/15-37
0x2_4208–0x2_42AC	Reserved	—	—	—
0x2_42B0	OSTBD—Out-of-sequence TxBD register	R/W	0x0800_0000	15.5.3.3.8/15-37
0x2_42B4	OSTBDP—Out-of-sequence Tx data buffer pointer register	R/W	0x0000_0000	15.5.3.3.9/15-39
0x2_42B8–0x2_42FC	Reserved	—	—	—

Table 15-3. Module Memory Map (continued)

Offset	Name	Access[1]	Reset	Section/Page
TSEC1 Receive Control and Status Registers				
0x2_4300	RCTRL—Receive control register	R/W	0x0000_0000	15.5.3.4.1/15-40
0x2_4304	RSTAT—Receive status register	R/W	0x0000_0000	15.5.3.4.2/15-41
0x2_4308	Reserved	—	—	—
0x2_430C	RBDLEN—RxBD data length register	R	0x0000_0000	15.5.3.4.3/15-41
0x2_4310	RXIC—Receive interrupt coalescing configuration register	R/W	0x0000_0000	15.5.3.4.4/15-42
0x2_4314–0x2_4320	Reserved	—	—	—
0x2_4324	CRBPTR—Current RxBD pointer register	R	0x0000_0000	15.5.3.4.5/15-43
0x2_4328–0x2_433C	Reserved	—	—	—
0x2_4340	MRBLR—Maximum receive buffer length register	R/W	0x0000_0000	15.5.3.4.6/15-43
0x2_4344–0x2_4380	Reserved	—	—	—
0x2_4384	RBPTR—RxBD pointer register	R/W	0x0000_0000	15.5.3.4.7/15-44
0x2_4388–0x2_4400	Reserved	—	—	—
0x2_4404	RBASE—RxBD base address register	R/W	0x0000_0000	15.5.3.4.8/15-44
0x2_4408–0x2_44FC	Reserved	—	—	—
TSEC1 MAC Registers				
0x2_4500	MACCFG1—MAC configuration register 1	R/W, R	0x0000_0000	15.5.3.6.1/15-48
0x2_4504	MACCFG2—MAC configuration register 2	R/W	0x0000_7000	15.5.3.6.2/15-49
0x2_4508	IPGIFG—Inter-packet gap/inter-frame gap register	R/W	0x4060_5060	15.5.3.6.3/15-51
0x2_450C	HAFDUP—Half-duplex register	R/W	0x00A1_F037	15.5.3.6.4/15-52
0x2_4510	MAXFRM—Maximum frame length register	R/W	0x0000_0600	15.5.3.6.5/15-53
0x2_4514–0x2_451C	Reserved	—	—	—
0x2_4520	MIIMCFG—MII management configuration register	R/W	0x0000_0000	15.5.3.6.6/15-53
0x2_4524	MIIMCOM—MII Management command register	R/W	0x0000_0000	15.5.3.6.7/15-54
0x2_4528	MIIMADD—MII Management address register	R/W	0x0000_0000	15.5.3.6.8/15-55
0x2_452C	MIIMCON—MII Management control register	W	0x0000_0000	15.5.3.6.9/15-56
0x2_4530	MIIMSTAT—MII Management status register	R	0x0000_0000	15.5.3.6.10/15-56
0x2_4534	MIIMIND—MII Management indicator register	R	0x0000_0000	15.5.3.6.11/15-57
0x2_4538	Reserved	—	—	—

Table 15-3. Module Memory Map (continued)

Offset	Name	Access[1]	Reset	Section/Page
0x2_453C	IFSTAT—Interface status register	R	0x0000_0000	15.5.3.6.12/15-58
0x2_4540	MACSTNADDR1—Station address register, part 1	R/W	0x0000_0000	15.5.3.6.13/15-58
0x2_4544	MACSTNADDR2—Station address register, part 2	R/W	0x0000_0000	15.5.3.6.14/15-59
0x2_4548–0x2_467C	Reserved	—	—	—
TSEC1 RMON MIB Registers				
TSEC1 Transmit and Receive Counters				
0x2_4680	TR64—Transmit and receive 64-byte frame counter register	R/W	0x0000_0000	15.5.3.7.1/15-60
0x2_4684	TR127—Transmit and receive 65- to 127-byte frame counter register	R/W	0x0000_0000	15.5.3.7.2/15-61
0x2_4688	TR255—Transmit and receive 128- to 255-byte frame counter register	R/W	0x0000_0000	15.5.3.7.3/15-61
0x2_468C	TR511—Transmit and receive 256- to 511-byte frame counter register	R/W	0x0000_0000	15.5.3.7.4/15-62
0x2_4690	TR1K—Transmit and receive 512- to 1023-byte frame counter register	R/W	0x0000_0000	15.5.3.7.5/15-62
0x2_4694	TRMAX—Transmit and receive 1024- to 1518-byte frame counter register	R/W	0x0000_0000	15.5.3.7.6/15-63
0x2_4698	TRMGV—Transmit and receive 1519- to 1522-byte good VLAN frame count register	R/W	0x0000_0000	15.5.3.7.7/15-63
TSEC1 Receive Counters				
0x2_469C	RBYT—receive byte counter register	R/W	0x0000_0000	15.5.3.7.8/15-64
0x2_46A0	RPKT—receive packet counter register	R/W	0x0000_0000	15.5.3.7.9/15-64
0x2_46A4	RFCS—receive FCS error counter register	R/W	0x0000_0000	15.5.3.7.10/15-65
0x2_46A8	RMCA—receive multicast packet counter register	R/W	0x0000_0000	15.5.3.7.11/15-65
0x2_46AC	RBCA—receive broadcast packet counter register	R/W	0x0000_0000	15.5.3.7.12/15-66
0x2_46B0	RXCF—receive control frame packet counter register	R/W	0x0000_0000	15.5.3.7.13/15-66
0x2_46B4	RXPF—receive PAUSE frame packet counter register	R/W	0x0000_0000	15.5.3.7.14/15-67
0x2_46B8	RXUO—receive unknown OP code counter register	R/W	0x0000_0000	15.5.3.7.15/15-67
0x2_46BC	RALN—receive alignment error counter register	R/W	0x0000_0000	15.5.3.7.16/15-68
0x2_46C0	RFLR—receive frame length error counter register	R/W	0x0000_0000	15.5.3.7.17/15-68
0x2_46C4	RCDE—receive code error counter register	R/W	0x0000_0000	15.5.3.7.18/15-69
0x2_46C8	RCSE—receive carrier sense error counter register	R/W	0x0000_0000	15.5.3.7.19/15-69
0x2_46CC	RUND—receive undersize packet counter register	R/W	0x0000_0000	15.5.3.7.20/15-70
0x2_46D0	ROVR—receive oversize packet counter register	R/W	0x0000_0000	15.5.3.7.21/15-70

Table 15-3. Module Memory Map (continued)

Offset	Name	Access[1]	Reset	Section/Page
0x2_46D4	RFRG—receive fragments counter register	R/W	0x0000_0000	15.5.3.7.22/15-71
0x2_46D8	RJBR—receive jabber counter register	R/W	0x0000_0000	15.5.3.7.23/15-71
0x2_46DC	RDRP—receive drop register	R/W	0x0000_0000	15.5.3.7.24/15-72
TSEC1 Transmit Counters				
0x2_46E0	TBYT—Transmit byte counter register	R/W	0x0000_0000	15.5.3.7.25/15-72
0x2_46E4	TPKT—Transmit packet counter register	R/W	0x0000_0000	15.5.3.7.26/15-73
0x2_46E8	TMCA—Transmit multicast packet counter register	R/W	0x0000_0000	15.5.3.7.27/15-73
0x2_46EC	TBCA—Transmit broadcast packet counter register	R/W	0x0000_0000	15.5.3.7.28/15-74
0x2_46F0	TXPF—Transmit PAUSE control frame counter register	R/W	0x0000_0000	15.5.3.7.29/15-74
0x2_46F4	TDFR—Transmit deferral packet counter register	R/W	0x0000_0000	15.5.3.7.30/15-75
0x2_46F8	TEDF—Transmit excessive deferral packet counter register	R/W	0x0000_0000	15.5.3.7.31/15-75
0x2_46FC	TSCL—Transmit single collision packet counter register	R/W	0x0000_0000	15.5.3.7.32/15-76
0x2_4700	TMCL—Transmit multiple collision packet counter register	R/W	0x0000_0000	15.5.3.7.33/15-76
0x2_4704	TLCL—Transmit late collision packet counter register	R/W	0x0000_0000	15.5.3.7.34/15-77
0x2_4708	TXCL—Transmit excessive collision packet counter register	R/W	0x0000_0000	15.5.3.7.35/15-77
0x2_470C	TNCL—Transmit total collision counter register	R/W	0x0000_0000	15.5.3.7.36/15-78
0x2_4710	Reserved	—	—	—
0x2_4714	TDRP—Transmit drop frame counter register	R/W	0x0000_0000	15.5.3.7.37/15-78
0x2_4718	TJBR—Transmit jabber frame counter register	R/W	0x0000_0000	15.5.3.7.38/15-79
0x2_471C	TFCS—Transmit FCS error counter register	R/W	0x0000_0000	15.5.3.7.39/15-79
0x2_4720	TXCF—Transmit control frame counter register	R/W	0x0000_0000	15.5.3.7.40/15-80
0x2_4724	TOVR—Transmit oversize frame counter register	R/W	0x0000_0000	15.5.3.7.41/15-80
0x2_4728	TUND—Transmit undersize frame counter register	R/W	0x0000_0000	15.5.3.7.42/15-81
0x2_472C	TFRG—Transmit fragments frame counter register	R/W	0x0000_0000	15.5.3.7.43/15-81
TSEC1 General Registers				
0x2_4730	CAR1—Carry register one register	R	0x0000_0000	15.5.3.7.44/15-82
0x2_4734	CAR2—Carry register two register	R	0x0000_0000	15.5.3.7.45/15-83
0x2_4738	CAM1—Carry register one mask register	R/W	0xFE01_FFFF	15.5.3.7.46/15-85
0x2_473C	CAM2—Carry register two mask register	R/W	0x000F_FFFF	15.5.3.7.47/15-86
0x2_4740– 0x2_47FC	Reserved	—	—	—

Table 15-3. Module Memory Map (continued)

Offset	Name	Access[1]	Reset	Section/Page
TSEC1 Hash Function Registers				
0x2_4800	IADDR0—Individual address register 0	R/W	0x0000_0000	15.5.3.8.1/15-87
0x2_4804	IADDR1—Individual address register 1	R/W	0x0000_0000	
0x2_4808	IADDR2—Individual address register 2	R/W	0x0000_0000	
0x2_480C	IADDR3—Individual address register 3	R/W	0x0000_0000	
0x2_4810	IADDR4—Individual address register 4	R/W	0x0000_0000	
0x2_4814	IADDR5—Individual address register 5	R/W	0x0000_0000	
0x2_4818	IADDR6—Individual address register 6	R/W	0x0000_0000	
0x2_481C	IADDR7—Individual address register 7	R/W	0x0000_0000	
0x2_4820–0x2_487C	Reserved	—	—	—
0x2_4880	GADDR0—Group address register 0	R/W	0x0000_0000	15.5.3.8.2/15-88
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	—	—	—
TSEC1 Attribute Registers				
0x2_4BF8	ATTR—Attribute register	R	0x0000_0000	15.5.3.9.1/15-89
0x2_4BFC	Reserved	—	—	—
TSEC1 Future Expansion Space				
0x2_4C00–0x2_4FFF	Reserved	—	—	—
TSEC2 Registers				
0x2_5000–0x2_5FFF	TSEC2 registers NOTE: TSEC2 has the same memory-mapped registers that are described for TSEC1 from 0x 2_4000 to 0x2_4FFF except the offsets are from 0x 2_5000 to 0x2_5FFF.			

[1] R = read-only, W = write only, R/W = read and write, LH = latches high, SC = self-clearing,

15.5.3 Memory-Mapped Register Descriptions

This section provides a detailed description of all the TSEC registers. Because all of the TSEC registers are 32 bits wide, only 32-bit register accesses are supported.

15.5.3.1 TSEC 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.

15.5.3.1.1 Interrupt Event Register (IEVENT)

If an event occurs that sets a bit in the interrupt event (IEVENT) register, shown in Figure 15-8, an interrupt is generated if the corresponding bit in the interrupt enable register (IMASK) is also set. Clearing the IEVENT bit clears the interrupt signal. The bit in the IEVENT register is cleared if a 1 is written to that bit position. A write of 0 has no effect.

These interrupts can be divided into operational interrupts, transceiver/network error interrupts, and internal error interrupts. Interrupts that may occur in normal operation are:

- GTSC, GRSC, TXF, TXB, TXC, RXF, RXB, RXC, MSRO, MMRD, and MMWR

Interrupts resulting from errors/problems detected in the network or transceiver are as follows:

- BABR, BABT, LC, and CRL/XDA

Interrupts resulting from internal errors are as follows:

- EBERR, XFUN, and BSY

Some of the error interrupts are independently counted in the management information base (MIB) block counters. Software may choose to mask off these interrupts because these errors are visible to network management via the MIB counters.

Offset TSEC1:0x2_4010; TSEC2: 0x2_5010 — Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BABR	RXC	BSY	EBERR	—	MSRO	GTSC	BABT	TXC	TXE	TXB	TXF	—	LC	CRL/ XDA	XFUN
Reset	All zeros															

	16	17	20	21	22	23	24	25	31
R W	RXB	—		MMRD	MMWR	GRSC	RXF	—	
Reset	All zeros								

Figure 15-6. IEVENT Register Definition

Table 15-4 describes the fields of the IEVENT register.

Table 15-4. IEVENT Field Descriptions

Bits	Name	Description
0	BABR	Babbling receive error. Indicates whether a frame was received with length in excess of the MAC's maximum frame length register while MACCFG2[HFE] is set. 0 Excessive frame not received. 1 Excessive frame received.
1	RXC	Receive control interrupt. Dictates whether a control frame was received. As soon as the transmitter finishes sending the current frame, a pause operation is performed if MACCFG1[Rx_Flow] is set. 0 Control frame not received. 1 Control frame received.
2	BSY	Busy condition interrupt. Indicates whether a frame was received and discarded due to a lack of buffers. When IEVENT[BSY] is set RSTAT[QHLT] is also set. In order to begin receiving packets again, the user must clear RSTAT[QHLT]. This bit and RSTAT[QHLT] are set whenever the TSEC reads an RxBD with its Empty field cleared. 0 No frame received and discarded. 1 Frame received and discarded.
3	EBERR	Ethernet bus error. Indicates whether a coherency system bus (CSB) error for a memory read occurred while a DMA transaction was underway. If the EBERR is set while transmission is in progress, the DMA stops sending data to the Tx FIFO which eventually causes an underrun error (XFUN). If the EBERR is set while receiving a frame, the DMA discards the frame. 0 No CSB error occurred. 1 CSB error occurred.
4	—	Reserved
5	MSRO	MSTAT register overflow. This interrupt is asserted if the count for one of the MSTAT registers has exceeded the size of the register. 0 MSTAT count not exceeding register size. 1 MSTAT count exceeds register size.
6	GTSC	Graceful transmit stop complete. Graceful stop means that the transmitter is put into a pause state after completion of the frame currently being transmitted. This interrupt is asserted for one of two reasons. • A graceful stop, which was initiated by setting DMACTRL[GTS], is now complete. • A graceful stop, which was initiated by setting TCTRL[TFC_PAUSE], is now complete. 0 Graceful transmit stop not complete or not initiated. 1 Graceful transmit stop completed.
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[HFE] is cleared. Frame truncation occurs when this condition occurs. TxBD[TXTRUNC] is set in the last TxBD (TxBD[L] is set) of the frame. 0 Transmitted frame length not exceeding maximum frame length. 1 Transmitted frame length exceeding maximum frame length.
8	TXC	Transmit control interrupt. This bit indicates whether a control frame was transmitted. 0 Control frame not transmitted. 1 Control frame transmitted.

Table 15-4. IEVENT Field Descriptions (continued)

Bits	Name	Description
9	TXE	Transmit error. This bit indicates whether an error occurred on the transmitted channel that has caused TSTAT[THLT] to be set by TSEC. This bit is set whenever any transmit error occurs that causes the transmitter to halt (EBERR, LC, CRL/XDA, XFUN). It is not set if DMACTRL[WOP] is set and TSEC runs out of TxBDs to process. In order to begin transmitting packets again, the user must clear TSTAT[THLT]. 0 No transmit channel error occurred. 1 Transmit channel error occurred.
10	TXB	Transmit buffer. This bit indicates whether 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.
11	TXF	Transmit frame interrupt. This bit indicates whether 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. 0 No frame transmitted/TxBD not updated. 1 Frame transmitted/TxBD updated.
12	—	Reserved
13	LC	Late collision. This bit indicates whether 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/ XDA	Collision retry limit/excessive defer abort. This bit indicates whether one of two conditions occurred while attempting to transmit a frame: 1) the number of successive transmission collisions has exceeded the MAC's HAFDUP[Retransmission Maximum] count or 2) an excessive defer abort condition has occurred. An excessive defer abort condition occurs when the TSEC waits more than 3036 bytes while attempting to send a frame and HAFDUP[EXCESS_DEFER] is 0. The TxBD[DEF] or OSTBD[DEF] is also set. In either case the frame is discarded without being transmitted and the TSEC is halted (TSTAT[THLT] is set). The CRL or XDA condition can only occur while in half-duplex mode. 0 Successive transmission collisions do not exceed maximum and/or no excessive defer abort condition has occurred. 1 Successive transmission collisions exceed maximum or an excessive defer abort condition has occurred.
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. These bits indicate whether 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.

Table 15-4. IEVENT Field Descriptions (continued)

Bits	Name	Description
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 bit is said 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. Indicates whether the last receive buffer descriptor (RxBD) of a frame was updated. This occurs only if the I (interrupt) bit in the buffer descriptor status word is set. 0 Frame not received. 1 Frame received.
25–31	—	Reserved

15.5.3.1.2 Interrupt Mask Register (IMASK)

IMASK, shown in Figure 15-7, provides control over which possible interrupt events are allowed to generate an actual interrupt. All implemented bits in this CSR are R/W. This register is cleared upon a hardware reset. If the corresponding bits in both the IEVENT and IMASK registers are set, an interrupt is generated. The interrupt signal can be cleared by clearing the corresponding IEVENT bit.

Offset TSEC1:0x2_4014; TSEC2:0x2_5014 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BREN	RXCEN	BSYEN	EBERREN	—	MSROEN	GTSCEN	BTEN	TXCEN	TXEEN	TXBEN	TXFEN	—	LCEN	CRL/ XDAEN	XFUNEN
Reset	All zeros															

	16	17	20	21	22	23	24	25	31
R W	RXBEN	—		MMRDE	MMWR	GRSCEN	RXFEN	—	
Reset	All zeros								

Figure 15-7. IMASK Register Definition

Table 15-5 describes the fields of the IMASK register.

Table 15-5. 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	MSTAT register 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
13	LCEN	Late collision enable
14	CRL/XDAEN	Collision retry limit/excessive defer enable
15	XFUNEN	Transmit FIFO underrun enable
16	RXBEN	Receive buffer interrupt enable
17–20	—	Reserved
21	MMRD	MII management read completion enable
22	MMWR	MII management write completion enable
23	GRSCEN	Graceful receive stop complete interrupt enable
24	RXFEN	Receive frame interrupt enable
25–31	—	Reserved

15.5.3.1.3 Error Disabled Register (EDIS)

The error disabled register, shown in Figure 15-8, controls error reporting by the TSEC. The IEVENT bit corresponding to an error is not set if the error is disabled in EDIS.

Offset TSEC1:0x2_4018; TSEC2:0x2_5018 Access: Read/Write

	0	1	2	3	4	8	9	10	12	13	14	15
R W	—		BSYDIS	EBERRDIS	—		TXEDIS	—		LCDIS	CRL/ XDADIS	XFUNDIS
Reset	All zeros											

	16	31
R W	—	
Reset	All zeros	

Figure 15-8. Error Disabled Register (EDIS)

Table 15-6 describes the fields of the EDIS register.

Table 15-6. EDIS Field Descriptions

Bits	Name	Description
0–1	—	Reserved
2	BSYDIS	Busy disable 0 Allow TSEC 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 TSEC 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.
4–8	—	Reserved
9	TXEDIS	Transmit error disable 0 Allow TSEC 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 TSEC 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.

Table 15-6. EDIS Field Descriptions (continued)

Bits	Name	Description
14	CRL/XDA DIS	Collision retry limit excessive defer abort disable 0 Allow TSEC to report IEVENT[CRL/XDA] status, set the buffer descriptor RL field, and halt buffer descriptor queue if CRL/XDA condition occurs. 1 Do not set IEVENT[CRL/XDA] nor the buffer descriptor RL field, and do not halt buffer descriptor queue if CRL/XDA condition occurs.
15	XFUNDIS	Transmit FIFO underrun disable 0 Allow TSEC 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.
16–31	—	Reserved

15.5.3.1.4 Ethernet Control Register (ECNTRL)

ECNTRL, shown in Figure 15-9, is used to reset, configure, and initialize the TSEC.

Offset TSEC1:0x2_4020; TSEC2:0x2_5020 Access: Read/Write

	0														15
R	—														
W															
Reset	All zeros														

	16	17	18	19	20	25	26	27	28	29	31
R	—	CLRCNT	AUTOZ	STEN	—		TBIM	RPM	R100M	—	
W											
Reset	All zeros										

Figure 15-9. ECNTRL Register Definition

Table 15-7 describes the fields of the ECNTRL register.

Table 15-7. ECNTRL Field Descriptions

Bits	Name	Description
0–16	—	Reserved
17	CLRCNT	Clear all statistics counters 0 Allow MSTAT counters to continue to increment. 1 Reset all MSTAT counters. This bit is self-resetting.
18	AUTOZ	Automatically zero addressed statistical counter values, input to MSTAT module. 0 The user must write the addressed counter zero after a host read. 1 The addressed counter value is zeroed on 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 15-7. ECNTRL Field Descriptions (continued)

Bits	Name	Description
19	STEN	Statistics enabled, input to MSTAT module. 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.
20–25	—	Reserved
26	TBIM	Ten-bit interface mode. If this bit is set, ten-bit interface mode is enabled. This bit can be configured by the reset configuration high word, TSEC1M and TSEC2M fields. 0 GMII or MII mode interface 1 TBI mode interface
27	RPM	Reduced signal mode. If this bit is set, a reduced signal interface is expected. ECNTRL[TBI Mode] selects GMII or TBI. If RPM is selected, the MAC must be in GMII or TBI mode (RMII is not supported). This register can be configured by the reset configuration high word, TSEC1M and TSEC2M fields. 0 GMII or TBI in non-reduced signal mode configuration 1 RGMII or RTBI reduced signal mode
28	R100M	RGMII 100 mode. This bit is ignored unless RPM is set and MACCFG2[I/F Mode] is assigned to 10/100 (01). If this bit is set, the TSEC interface is in 100 Mbps speed. 0 RGMII is in 10 Mbps mode 1 RGMII is in 100 Mbps mode
29–31	—	Reserved

15.5.3.1.5 Minimum Frame Length Register (MINFLR)

MINFLR, shown in Figure 15-10, is written by the user. It tells TSEC the smallest packet to accept and place in a receive buffer pointed to by the RxBD.

Offset TSEC1:0x2_4024; TSEC2:0x2_5024 Access: Read/Write

	0																						24	25						31
R W	—																							MINFLR						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

Figure 15-10. MINFLR Register Definition

Table 15-8 describes the fields of the MINFLR register.

Table 15-8. MINFLR Field Descriptions

Bits	Name	Description
0–24	—	Reserved
25–31	MINFLR	Minimum receive frame length (typically 64 decimal). If the Ethernet receives an incoming frame shorter than MINFLR, it discards that frame unless RCTRL[RSF] (receive short frames) is set, in which case RxBD[SH] (frame too short) is set in the last RxBD. The largest allowable value for MINFLR is 64. Unlike the MPC8260, in which PADs are added to make the transmit frame equal to MINFLR bytes, if padding is requested, TSEC always PADS transmit frames to 64 bytes ignoring MINFLR. MINFLR is only used to determine the minimum size of acceptable receive frames.

15.5.3.1.6 Pause Time Value Register (PTV)

PTV, shown in Figure 15-11, is written by the user to store the pause duration used when the TSEC initiates a pause frame via 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 15.6.2.7, “Flow Control,” for additional details.

Offset TSEC1:0x2_4028;
TSEC2:0x2_5028;

Access: Read/Write

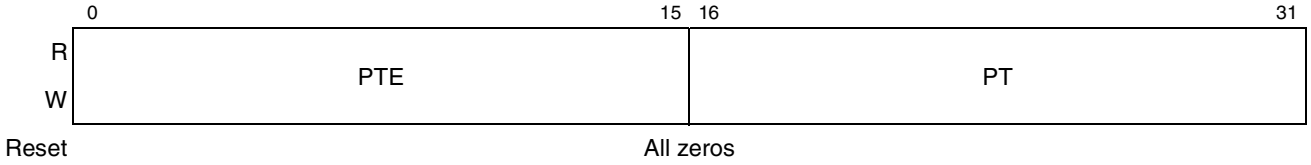


Figure 15-11. PTV Register Definition

Table 15-9 describes the fields of the PTV register.

Table 15-9. 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 (i.e. 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 15.6.2.7, “Flow Control.”

15.5.3.1.7 DMA Control Register (DMACTRL)

DMACTRL, shown in Figure 15-12, is used to configure the DMA block register.

Offset TSEC1:0x2_402C; TSEC2:0x2_502C — Access: Read/Write

	0–23	24	25	26	27	28	29	30	31
R W	—	TDSEN	TBDSEN	—	GRS	GTS	—	WWR	WOP
Reset	All zeros								

Figure 15-12. DMACTRL Register Definition

Table 15-10 describes the fields of the DMACTRL register.

Table 15-10. DMACTRL Field Descriptions

Bits	Name	Description
0–23	—	Reserved
24	TDSEN	Tx data snoop enable 0 Disables snooping of all transmit frames from memory. 1 Enables snooping of all transmit frames from memory.
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 buffer of the receive frame associated with the EOF is closed and the IEVENT[GRSC] is set. Because the MAC's receive enable bit (MACCFG[Rx_EN]) may still be set, the MAC may continue to receive but the TSEC ignores the receive data until GRS is cleared. If this bit is cleared, the TSEC 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 bit in IEVENT[GRSC] 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 TSEC hardware without fear of conflict. 0 TSEC scans input data stream for valid frame. 1 TSEC stops receiving frames following completion of current frame.
28	GTS	Graceful transmit stop. If this bit is set, the Ethernet controller stops transmission after the completion of any frame that is currently being transmitted and the GTSC interrupt in the IEVENT register is asserted. If frame transmission is not currently underway, the GTSC interrupt is asserted immediately. Once transmission has completed, a "restart" can be accomplished by clearing GTS. 0 Controller continues. 1 Controller stops transmission after completion of current frame.
29	—	Reserved
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, IE, XFUN, LC, CRL/XDA, RXB, RXF, the TSEC waits for acknowledgement from system that the transmit or receive BD being updated was stored in memory.

Table 15-10. DMACTRL Field Descriptions (continued)

Bits	Name	Description
31	WOP	Wait or poll. This bit, which is applicable only to the transmitter, provides the user the option for the TSEC to periodically poll TxBD or to wait for software to tell TSEC to fetch a buffer descriptor. While operating in the wait mode, the TSEC allows two additional reads of a descriptor which is not ready before entering a halt state. No interrupt is driven. (IEVENT[TXE] is clear.) To resume transmission, software must clear TSTAT[THLT]. Note that if this bit is set, the user must ensure that all TxBDs involved with sending a frame have their ready bits set before any transmission begins. Otherwise, TSEC behaves in a boundedly undefined fashion. A buffer descriptor is considered not ready when its TxBD[Ready] field is clear or its TxBD[Data Length] field is zero. It is considered ready when both TxBD[Ready] is set and TxBD[Data Length] is non-zero. In wait mode (DMACTRL[WOP] is set) when TSEC is processing a frame and an intermediate TxBD's ready bit is cleared, TSEC does not halt, but instead continuously polls the same TxBD until the TxBD becomes ready or an Ethernet interface error or a memory error is encountered. If the TxBD becomes ready, TSEC continues to process the frame. If an error occurs before the TxBD becomes ready, TSEC reports the error in both the IEVENT register as well as the TxBD for Ethernet interface errors, or the IEVENT[EBERR] for a memory error and sets the TSTAT[THLT] bit. Note that software must eventually set all of its TxBDs for a frame, because TSEC continuously reads an intermediate TxBD until it becomes ready if insufficient data has been read to surpass the FIFO Transmit Threshold (FIFO_TX_THR) register value. Note, the polling in Wait mode differs from the polling in Poll mode. In Poll mode polling is performed every 512 Ethernet bus Tx clocks. In Wait mode polling is performed as fast as possible after the TxBD[Ready]=0 is read. If software is slow in setting all TxBD Ready bits of a frame to a one, then the system performance may decrease. 0 Poll TxBD every 512 clocks. 1 Do not poll, but wait for a write to TSTAT[THLT].

15.5.3.1.8 TBI Physical Address Register (TBIPA)

TBIPA, shown in Figure 15-15, assigns a physical address to the TBI for MII management configuration. The TBI registers are accessed at the offset of TBIPA. For detailed descriptions of the TBI registers (the MII register set for the ten-bit interface), see Section 15.5.4, "Ten-Bit Interface (TBI)."

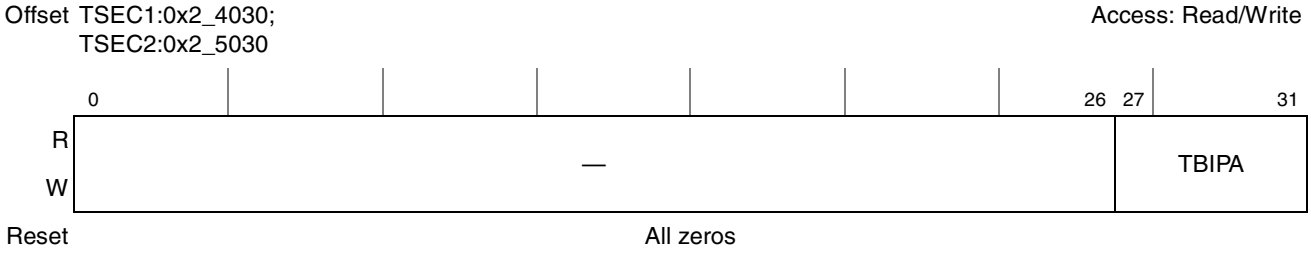


Figure 15-13. TBIPA Register Definition

Table 15-11 describes the fields of the TBIPA register.

Table 15-11. TBIPA Field Descriptions

Bits	Name	Description
0–26	—	Reserved
27–31	TBIPA	TBI PHY address. This field is used to program the PHY address of the ten-bit interface's MII management bus. To access the TBI register the user must write the TBIPA value to the MIIMADD [PHY Address] register located in the MAC register section. Refer to Section 15.5.3.6.8, "MII Management Address Register (MIIMADD)."

15.5.3.2 TSEC FIFO Control and Status Registers

The TSEC FIFO control and status registers allow the user to change some of the default settings in the FIFO that can be used to optimize operation for performance or for safety. They must be set carefully in order to avoid an underrun condition. Underrun is an error condition in which data is not retrieved from external memory quickly enough, leaving the TX FIFO empty before the complete frame is transmitted. Because different combinations of events, several of which are determined by the user, can lead to underrun, the TSEC provides several FIFO registers that allow the user to select the proper setting to be able to tune the system and obtain the maximum performance with minimal chance of underrun. The principal causes for underrun in the TSEC are as follows:

- Misaligned data buffer addresses
- Small data buffer sizes
- Combinations of the above

It is recommended that the minimum size data buffers be 64 bytes and that data buffers be 64-byte aligned. The user can deviate from these recommended values to try to increase performance or to use less memory, but unless the default values of some of the FIFO registers are adjusted, the probability of an underrun may also increase. The FIFO_TX_THR (default is 256 entries or 1 Kbytes) indicates the amount of data required to be in the FIFO before starting the transmission of a frame. The FIFO_TX_STARVE (default is 128 entries or 512 bytes) is used to indicate that the amount of data in the FIFO is so low that the risk of underrun is extremely high. The FIFO_TX_STARVE_SHUTOFF (default is 256 entries or 1 Kbyte) contains the watermark level to be used for exiting the starve state. These registers are intended to allow the user to make the proper trade-off. If triggered, the starve mode, for instance, automatically raises the priority of TSEC fetches from memory.

15.5.3.2.1 FIFO Transmit Threshold Register (FIFO_TX_THR)

The main purpose of the threshold register is to trigger the unloading of FIFO data to the PHY. It represents the numerical SRAM entry (0-511 for 2-Kbyte FIFO) to trigger the threshold function. If the number of valid entries in the FIFO is equal to or greater than the threshold register, transmission can begin. This register is read/write by software and is initialized to 0000_0000_0000_0000_0000_0001_0000_0000 at system reset. Figure 15-14 shows FIFO_TX_THR.

Offset TSEC1:0x2_408C; TSEC2:0x2_508C — Access: Read/Write

	0																				22	23								31
R W	—																					FIFO_TX_THR								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-14. FIFO_TX_THR Register Definition

Table 15-12 describes the FIFO_TX_THR fields.

Table 15-12. FIFO_TX_THR Field Descriptions

Bits	Name	Description
0–22	—	Reserved
23–31	FIFO_TX_THR	FIFO transmit threshold. These bits mark the number of entries in the transmit FIFO that, if reached, trigger the unloading of frame data into the MAC.

15.5.3.2.2 FIFO Transmit Space Available Register (FIFO_TX_SP)

FIFO_TX_SP contains the exact number of entries that once the transmit used entry count is equal to or greater than the transmit space available, an indication is sent to the DMA controller.

Offset TSEC1:0x2_4094 (must be programmed to 0x0000_0025); TSEC2:0x2_5094 — Access: Read/Write

	0																				22	23								31
R W	—																					FIFO_TX_SP								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 15-15. FIFO_TX_SP Register Definition

Table 15-13. FIFO_TX_SP Field Descriptions

Bits	Name	Description
0–22	—	Reserved
23–31	FIFO_TX_SP	Transmit space available. Indicates the number of entries unused as a watermark level for the DMA controller. A status indicator is sent to the DMA controller whenever the FIFO transmit free entry count is greater than or equal to the FIFO_TX_SP.

15.5.3.2.3 FIFO Transmit Starve Register (FIFO_TX_STARVE)

FIFO_TX_STARVE, shown in Figure 15-16, informs the system of extremely imminent underrun conditions. It represents the numerical SRAM entry (0–511 for 2-Kbyte FIFO) to trigger the starve function. If the number of valid entries in the FIFO is less than or equal to the starve register, a starve alert is triggered.

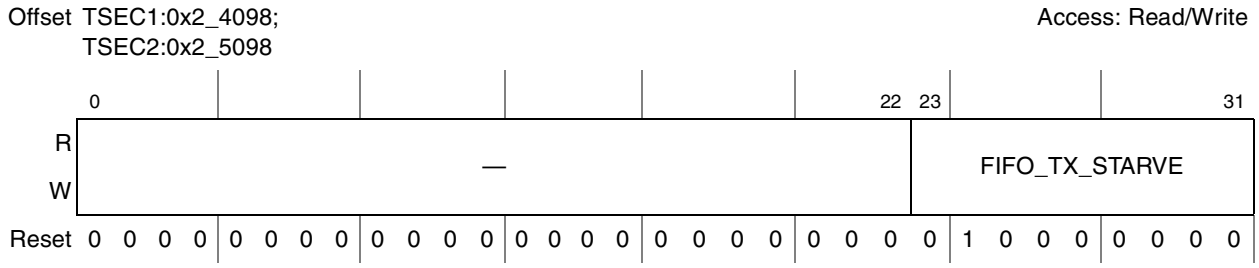


Figure 15-16. FIFO_TX_STARVE Register Definition

Table 15-14 describes the fields of the FIFO_TX_STARVE register.

Table 15-14. FIFO_TX_STARVE Field Descriptions

Bits	Name	Description
0–22	—	Reserved
23–31	FIFO_TX_STARVE	FIFO transmit starve. Indicates the value to trigger the transmit starve function. It triggers once the number of valid entries in the FIFO is less than or equal to the FIFO Tx starve. The starve state turns off if the number of valid entries in the FIFO becomes greater than or equal to the FIFO Tx starve shutoff register.

15.5.3.2.4 FIFO Transmit Starve Shutoff Register (FIFO_TX_STARVE_SHUTOFF)

The starve shutoff register, shown in Figure 15-17, contains the watermark level used for exiting the starve state. If the starve state is in effect and the number of valid entries in the FIFO becomes greater than or equal to the value in the FIFO transmit starve shutoff register, the starve condition ends.

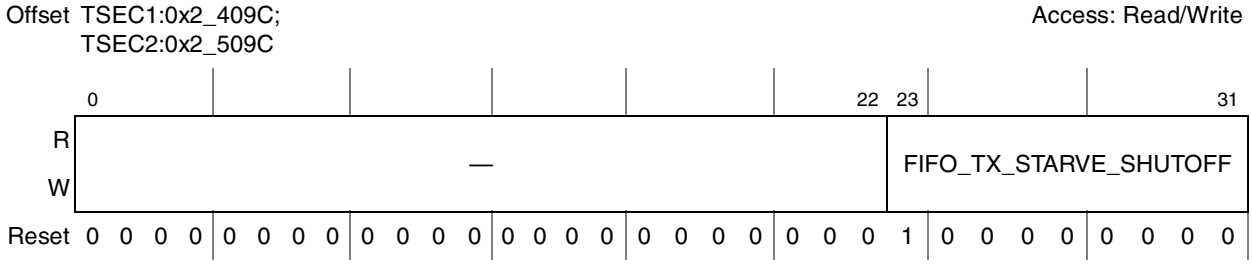


Figure 15-17. FIFO_TX_STARVE_SHUTOFF Register Definition

Table 15-15 describes the fields of the FIFO transmit starve shutoff register.

Table 15-15. FIFO_TX_STARVE_SHUTOFF Field Descriptions

Bits	Name	Description
0–22	—	Reserved
23–31	FIFO_TX_STARVE_ SHUTOFF	FIFO transmit starve shutoff. Indicates the value beyond which to exit the starve state. The starve state turns off if the number of valid entries in the FIFO becomes greater than or equal to the FIFO Tx starve shutoff register.

15.5.3.3 TSEC 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.

15.5.3.3.1 Transmit Control Register (TCTRL)

TCTRL, shown in Figure 15-18, is writable by the user to configure the transmit block.

Figure 15-18. TCTRL Register Definition

Table 15-16 describes the TCTRL fields.

Table 15-16. TCTRL Field Descriptions

Bits	Name	Description
0–19	—	Reserved, should be cleared
20	THDF	Transmit half-duplex flow control. Written by user. This bit is not self-resetting. 0 Disable back pressure. 1 Back pressure is applied to media.
21–26	—	Reserved, should be cleared
27	RFC_PAUSE	Receive flow control pause frame. Written by TSEC. 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 Pause duration in progress.
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 current transmission completes. Next, the GTSC interrupt in the IEVENT register is asserted. With transmission of data frames stopped, the MAC transmits a MAC control PAUSE frame with the duration value obtained from the PTV register. The TXC interrupt occurs after sending the control pause frame. Next, the MAC clears TFC_PAUSE and resumes transmitting data frames. Note that if the transmitter is paused due to user assertion of GTS or reception of a PAUSE frame, the MAC may still transmit a MAC control PAUSE frame. 0 No outstanding pause frame transmission request. 1 Pause frame transmission requested.
29–31	—	Reserved, should be cleared

15.5.3.3.2 Transmit Status Register (TSTAT)

TSTAT, shown in Figure 15-19, is written by TSEC to convey DMA status information.

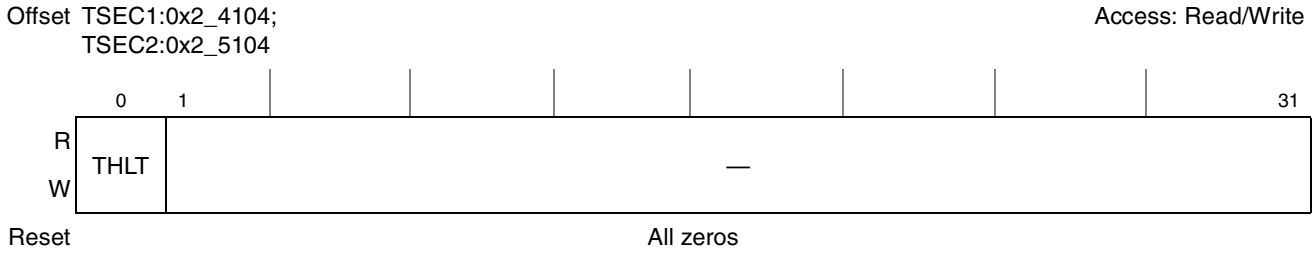


Figure 15-19. TSTAT Register Definition

Table 15-17 describes the fields of the TSTAT register.

Table 15-17. TSTAT Field Descriptions

Bits	Name	Description
0	THLT	Transmit halt 0 No hardware-initiated transmission halt. 1 TSEC transmission function halted. This bit is written by TSEC to inform the user that it is no longer processing transmit frames and that the transmit DMA function is disabled by hardware. To re-start the transmission function, the user must clear this bit by writing a one.
1–31	—	Reserved, should be cleared

15.5.3.3.3 TxBD Data Length Register (TBDLEN)

TBDLEN is a DMA register that contains the number of bytes remaining in the current transmit buffer. Figure 15-20 shows the TBDLEN register.

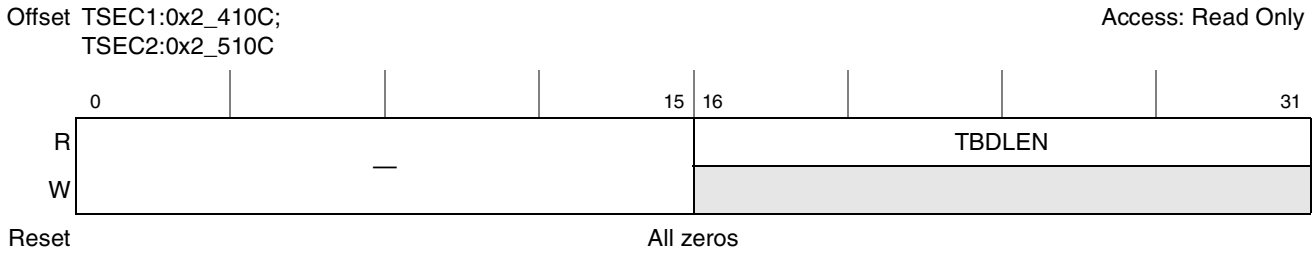


Figure 15-20. TBDLEN Register Definition

Table 15-18 describes the TBDLEN fields.

Table 15-18. TBDLEN Field Descriptions

Bits	Name	Description
0–15	—	Reserved, should be cleared
16–31	TBDLEN	Internally written by the DMA module. The transmit channel remains active until TBDLEN is 0.

15.5.3.3.4 Transmit Interrupt Coalescing Configuration Register (TXIC)

TXIC, shown in Figure 15-21, enables and configures the operational parameters for interrupt coalescing associated with transmitted frames. Refer to Section 15.6.2.8.1, "Interrupt Coalescing," for a functional description of interrupt coalescing.

Offset TSEC1:0x2_4110; TSEC2:0x2_5110 Access: Read/Write

	0	1	2	3	10	11	15	16	31
R	ICEN	—		ICFCT		—		ICTT	
W									
Reset	All zeros								

Figure 15-21. TXIC Register Definition

Table 15-18 describes the TXIC fields.

Table 15-19. TXIC Field Descriptions

Bits	Name	Description
0	ICEN	Interrupt coalescing enable 0 Interrupt coalescing is disabled. Interrupts are raised as they are received if the TSEC transmit frame interrupt is enabled (IMASK[TXFEN] is set). 1 Interrupt coalescing is enabled. If the TSEC transmit frame interrupt is enabled (IMASK[TXFEN] is set), an interrupt is raised when the threshold number of frames is reached (defined by TXIC[ICFCT]) or when the threshold timer expires (defined by TXIC[ICTT]).
1–2	—	Reserved, should be cleared
3–10	ICFCT	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[1]. Valid values for this field are from 1 to 255. A value of 0 is illegal. If set to 0, an interrupt can only be cleared by first clearing TXIC[ICEN] and then clearing the IEVENT[TXF] bit. Note that a value of 1 functionally defeats the advantages of interrupt coalescing since the frame threshold is reached with each frame transmitted.
11–15	—	Reserved, should be cleared
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[1], subject also to IMASK[TXFEN]. If frames have been transmitted but the frame count threshold has not been met, an interrupt is raised when the threshold timer expires. The threshold timer is reset once an interrupt has been asserted. It begins counting once the interrupt is cleared and IEVENT[TXF] is set. The threshold value is represented in units equal to 64 TSEC interface clocks. Valid values for this field are from 1 to 65535. A value of 0 is illegal and results in behavior identical to that when interrupt coalescing is disabled (TXIC[ICEN] is cleared).

[1] Interrupts resulting from the Interrupt bit (I) of the buffer descriptor in question and enabled subject to the IMASK register (IMASK[TXFEN] is set).

15.5.3.3.5 Current Transmit Buffer Descriptor Pointer Register (CTBPTR)

CTBPTR contains the address of the transmit buffer descriptor either currently being processed, or processed most recently. Figure 15-22 shows the CTBPTR register.

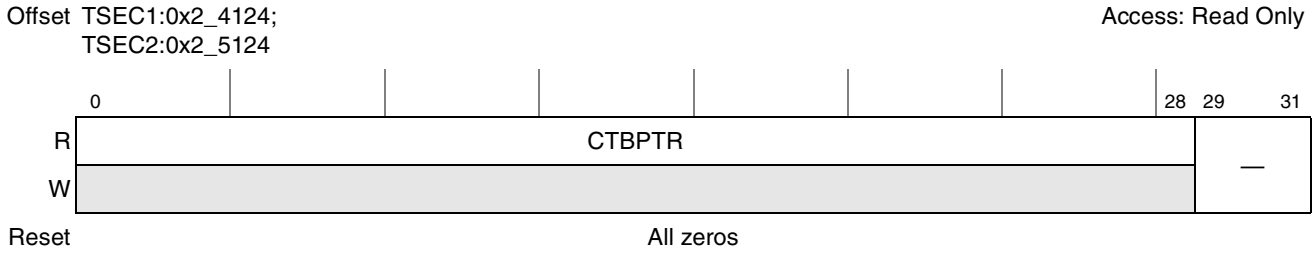


Figure 15-22. CTBPTR Register Definition

Table 15-20 describes the CTBPTR fields.

Table 15-20. CTBPTR Field Descriptions

Bits	Name	Description
0–28	CTBPTR	Internally written by the DMA module. The value increments by one (causing the register value to increment by eight) each time a descriptor is read from memory.
29–31	—	Reserved, should be cleared

15.5.3.3.6 Transmit Buffer Descriptor Pointer Register (TBPTR)

TBPTR, shown in Figure 15-23, contains the low-order 32 bits of the next transmit buffer descriptor address. TBPTR takes on the TBASE value when TBASE is written by software. Although not necessary in most applications, the user can modify this register when the transmitter has been gracefully stopped or halted, as indicated by IEVENT[GTSC] or TSTAT[THLT].

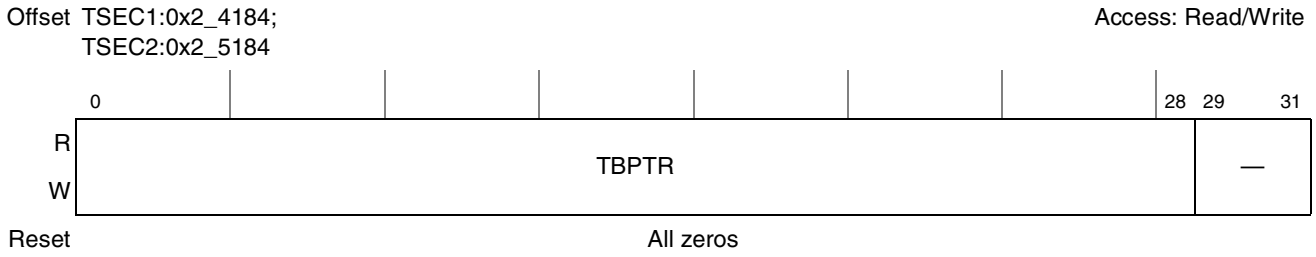


Figure 15-23. TBPTR Register Definition

Table 15-21 describes the fields of the TBPTR register.

Table 15-21. TBPTR Field Descriptions

Bits	Name	Description
0–28	TBPTR	Internally written by the DMA module. The value increments by one (causing the register value to increment by eight) each time a descriptor is read from memory.
29–31	—	Reserved, should be cleared

15.5.3.3.7 Transmit Descriptor Base Address Register (TBASE)

The TBASE register, shown in Figure 15-24, is written by the user with the TxBD base address. The value must be divisible by eight for the 8-byte data buffer descriptors.

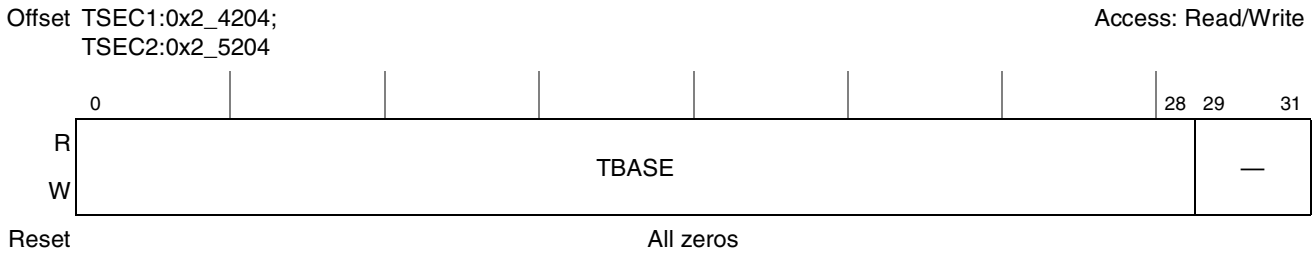


Figure 15-24. TBASE Register Definition

Table 15-22 describes the fields of the TBASE register.

Table 15-22. TBASE Field Descriptions

Bits	Name	Description
0–28	TBASE	Transmit base. TBASE defines the starting location in the memory map for the TSEC TxBDs. The base address 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 TSEC transmit function.
29–31	—	Reserved

15.5.3.3.8 Out-of-Sequence TxBD Register (OSTBD)

The out-of-sequence TxBD register, OSTBD, shown in Figure 15-25, includes the status/control and data length in the same format as a regular TxBD. It is useful for sending flow control frames. OSTBD[R] is always checked between frames. If it is not ready, a regular frame is sent. If a flow control frame is sent and OSTBD[I] is set, a TXC event is generated after frame transmission. This area must be cleared while not in use. Once the TSEC is in paused mode the out-of-sequence buffer descriptor cannot be used to send another flow control frame because the MAC regards it as a regular TxBD.

Offset TSEC1:0x2_42B0; TSEC2:0x2_52B0 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10–13	14	15	16–31
R/W	R	PAD/CRC	W	I	L	TC	DEF	TO1	HFE/LC	RL	RC	UN	—	OSTBDLEN
Reset	0	0	0	0	1	0	0	0	0	0	0 0 0 0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 15-25. OSTBD Register Definition

Table 15-23 describes the fields of the OSTBD register.

Table 15-23. OSTBD Field Descriptions

Bits	Name	Description
0	R	Ready. Written by TSEC 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 TSEC clears this bit after the buffer is transmitted or after an error condition is encountered. 1 The data buffer, which was 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 and CRC attachment for short frames. (Valid only when MACCFG2[PAD/CRC] is cleared, and MACCFG2[CRC EN] is cleared.) If MACCFG2[PAD/CRC] is set, pads are added to all short frames; however, this bit is ignored. 0 Do not add PADs to short frames unless OSTBD[TC] is set. 1 Add PADs to short frames. PAD bytes are inserted until the length of the transmitted frame equals 64 bytes. Unlike the MPC8260 which PADs up to MINFLR value, TSEC always PADs up to the IEEE minimum frame length of 64 bytes.
2	W	Wrap. Written by user. This bit is ignored by TSEC.
3	I	Interrupt. Written by user. 0 No interrupt is generated after this buffer is serviced. 1 IEVENT[TXF] is set after this buffer is serviced. This bit can cause an interrupt if IMASK[TXFEN] is enabled.
4	L	Last in frame. The OSTBD is always last in the frame, so L is always set (hardwired to a value of 1).
5	TC	Tx CRC. Written by user. (Valid only while it is set in the first BD and OSTBD[PAD/CRC] is cleared, MACCFG2[PAD/CRC] is cleared, and MACCFG2[CRC EN] is cleared.) If MACCFG2[PAD/CRC] is set or MACCFG2[CRC EN] is set, a CRC is added to all frames and this bit is ignored. 0 End transmission immediately after the last data byte, unless OSTBD[PAD/CRC] is set. 1 Transmit the CRC sequence after the last data byte.
6	DEF	Defer indication. Written by TSEC. Hardware updates this bit after transmitting a frame if used as a ‘Defer indicator.’ Software/user updates this bit while building a transmit buffer descriptor if used as a ‘Hardware Event indicator.’ 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	TO1	Transmit software ownership. This read/write bit may be used by software, as necessary. Its state does not affect the hardware nor is it affected by the hardware.

Table 15-23. OSTBD Field Descriptions (continued)

Bits	Name	Description
8	HFE/LC	Huge frame enable (written by user)/Late collision (written by TSEC) Huge frame enable. Written by user. Valid only while it is set in first BD and the 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 register. 1 Do not truncate the transmit frame. Late collision. Written by TSEC. 0 No late collision. 1 A collision occurred after 64 bytes are sent. The TSEC terminates the transmission and updates LC.
9	RL	Retransmission limit. Written by TSEC. 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 TSEC terminates the transmission and updates RL.
10–13	RC	Retry count. Written by TSEC. 0 The frame is sent correctly the first time. 1 More than zero attempts were needed to send the transmit frame. If this field is 15, 15 or more retries were needed. The Ethernet controller updates RC after sending the buffer.
14	UN	Underrun. Written by TSEC. 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. The TSEC terminates the transmission and updates UN.
15	—	Reserved
16–31	OSTBDLEN	Out-of-sequence TxBD data length. Written by user. Data length is the number of octets the TSEC transmits from this BD's data buffer. It is never modified by the TSEC. This field must be non-zero for a transfer to take place.

15.5.3.3.9 Out-of-Sequence Tx Data Buffer Pointer Register (OSTBDP)

OSTBDP, shown in Figure 15-26, contains the data buffer pointer fields in the same format as a regular TxBD, With OSTBD, it provides the complete 8-byte descriptor. This area must be cleared while not in use.

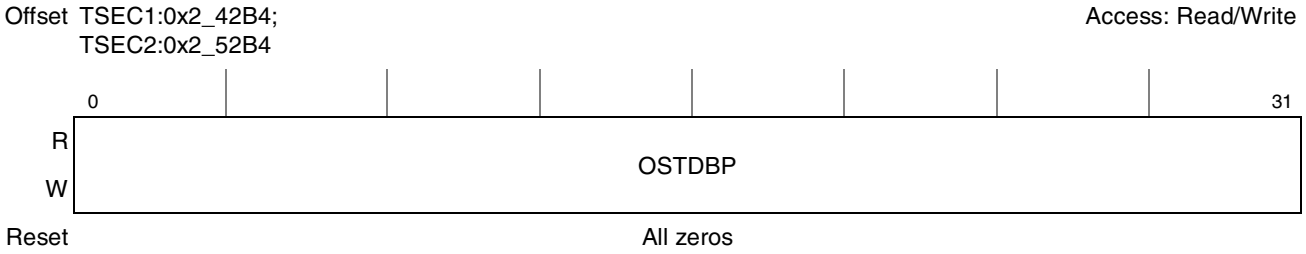


Figure 15-26. OSTDBP Register Definition

Table 15-24 describes OSTBDP.

Table 15-24. OSTBDP Field Descriptions

Bits	Name	Description
0–31	OSTDBP	Out-of-sequence Tx data buffer pointer. Written by user. The transmit data buffer pointer contains the address of the associated data buffer. There are no alignment requirements for this address.

15.5.3.4 TSEC 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.

15.5.3.4.1 Receive Control Register (RCTRL)

RCTRL, shown in Figure 15-27, controls the operational mode of the receive block. It must be written only after a system reset (at initialization) or if DMACTRL[GRS] is cleared.

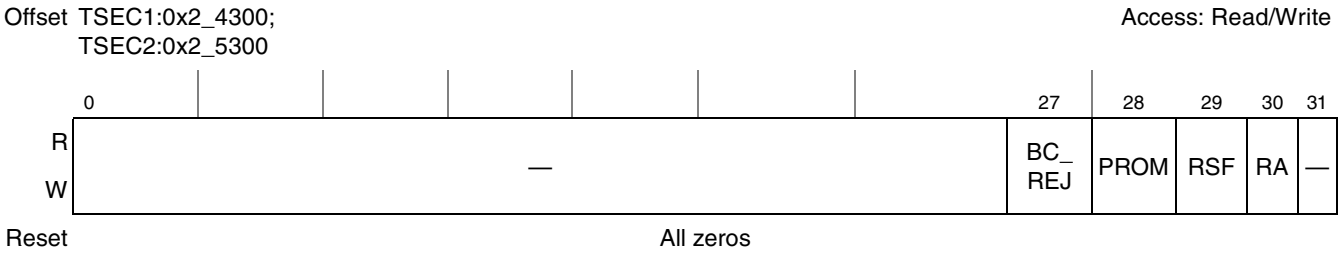


Figure 15-27. RCTRL Register Definition

Table 15-25 describes the RCTRL fields.

Table 15-25. RCTRL Field Descriptions

Bits	Name	Description
0–26	—	Reserved, should be cleared
27	BC_REJ	Broadcast frame reject. 0 Broadcast frame reject is disabled. 1 Broadcast frame reject is enabled. 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. 0 Promiscuous mode is disabled. 1 Promiscuous mode is enabled. When set, all frames except pause frames are accepted.
29	RSF	Receive short frame mode. 0 Receive short frame mode is disabled. 1 Receive short frame mode is enabled. When set, enables the reception of frames shorter than MINFLR bytes. Note that for short frames to be received when RSF is set, a DA hit must occur. When RSF is cleared, all frames shorter than MINFLR are automatically rejected.
30–31	—	Reserved, should be cleared

15.5.3.4.2 Receive Status Register (RSTAT)

TSEC writes to RSTAT, shown in Figure 15-28, under the following conditions:

- The receiver runs out of descriptors
- The receiver was halted because an error condition was encountered while receiving a frame

Software must clear the QHLT bit to take the TSEC receiver function out of a halt state.

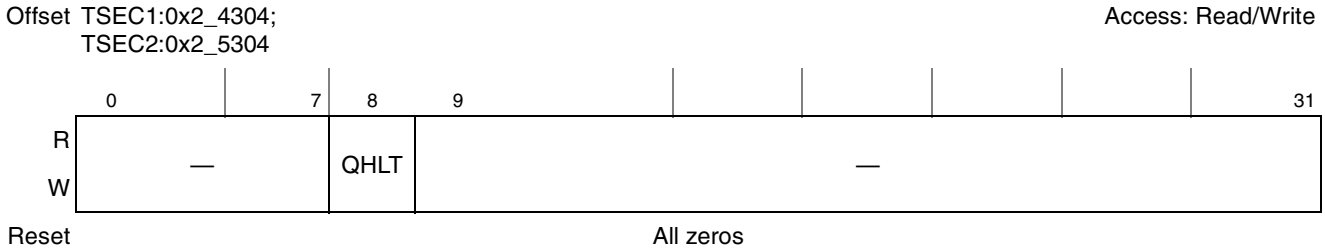


Figure 15-28. RSTAT Register Definition

Table 15-26 describes the fields of the RSTAT register.

Table 15-26. RSTAT Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8	QHLT	RxBD queue is halted. It is a hardware-initiated stop indication (DMA_CTRL[GRS] being set by the user does not cause this bit to be set.). The current frame and all other frames directed to the halted queue are discarded. A write with a value of 1 re-enables the queue for receiving. 0 RxBD queue is enabled for Ethernet reception. (That is, it is not halted.) 1 All Ethernet controller receive activity to RxBD queue is halted.
9–31	—	Reserved

15.5.3.4.3 RxBD Data Length Register (RBDLEN)

RBDLEN is a DMA register that contains the number of bytes remaining in the current receive buffer. Figure 15-29 shows the RBDLEN register.

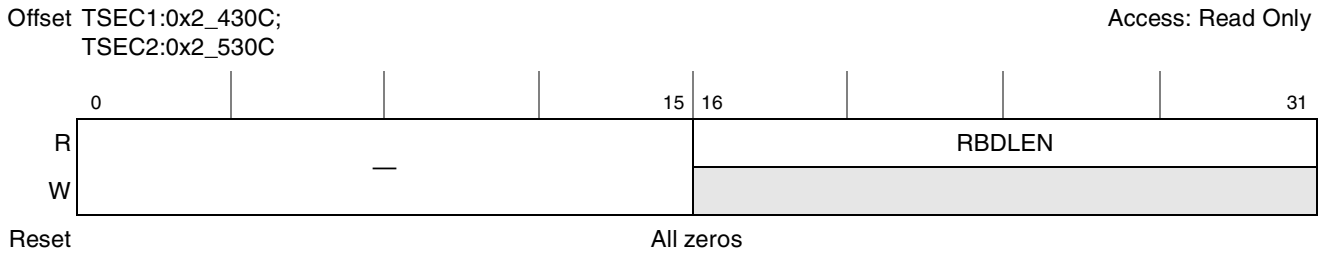


Figure 15-29. RBDLEN Register Definition

Table 15-27 describes the fields of the RBDLEN register.

Table 15-27. RBDLEN Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	RBDLEN	The RBDLEN is internally written by the DMA module. If RBDLEN is cleared, all activity in the receive channel stops.

15.5.3.4.4 Receive Interrupt Coalescing Configuration Register (RXIC)

RXIC enables and configures the operational parameters for interrupt coalescing associated with received frames. Section 15.6.2.8.1, "Interrupt Coalescing," provides additional details regarding the use of this register. Figure 15-30 shows the RXIC.

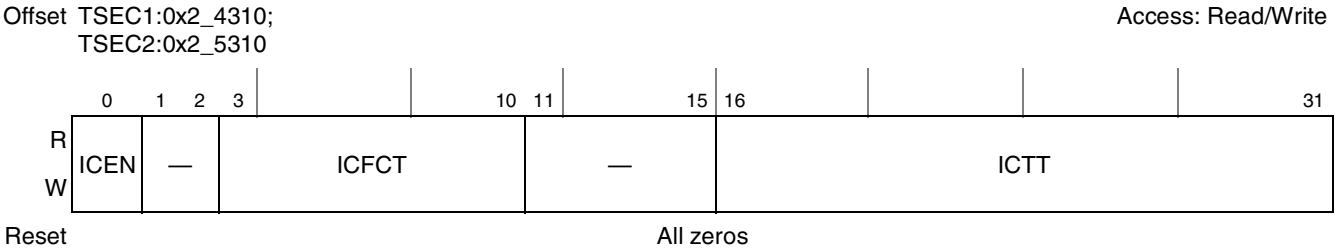


Figure 15-30. RXIC Register Definition

Table 15-18 describes the fields of the RXIC register.

Table 15-28. RXIC Field Descriptions

Bits	Name	Description
0	ICEN	Interrupt coalescing enable 0 Interrupt coalescing is disabled. Interrupts are raised as they are received if the TSEC receive frame interrupt is enabled (IMASK[RXFEN] is set). 1 Interrupt coalescing is enabled. If the TSEC receive frame interrupt is enabled (IMASK[RXFEN] is set), an interrupt is raised when the threshold number of frames is reached (defined by RXIC[ICFCT]) or when the threshold timer expires (defined by RXIC[ICTT]).
1–2	—	Reserved
3–10	ICFCT	Interrupt coalescing frame count threshold. While interrupt coalescing is enabled (RXIC[ICEN] is set), this value determines how many frames are received before raising an interrupt[1]. Valid values for this field are from 1 to 255. A value of 0 is illegal. If set to 0, an interrupt can only be cleared by first clearing RXIC[ICEN] and then clearing IEVENT[RXF]. Note that a value of 1 functionally defeats the advantages of interrupt coalescing since the frame threshold is reached with each frame received.
11–15	—	Reserved
16–31	ICTT	Interrupt coalescing timer threshold. While interrupt coalescing is enabled (RXIC[ICEN] is set), this value determines the maximum amount of time after receiving a frame before raising an interrupt[1], subject also to IMASK[RXFEN]. If frames have been received but the frame count threshold has not been met, an interrupt is raised when the threshold timer expires. The threshold timer is reset once an interrupt has been asserted. It begins counting once the interrupt is cleared and IEVENT[RXF] is set. The threshold value is represented in units equal to 64 TSEC interface clocks. Valid values for this field are from 1 to 65535. A value of 0 is illegal and results in behavior identical to that when interrupt coalescing is disabled (RXIC[ICEN] is cleared).

1 Interrupts resulting from the interrupt bit (I) of the buffer descriptor in question and enabled subject to the IMASK register (IMASK[RXFEN] is set).

15.5.3.4.5 Current Receive Buffer Descriptor Pointer Register (CRBPTR)

CRBPTR contains the address of the receive buffer descriptor either currently being processed, or processed most recently. Figure 15-31 shows the CRBPTR register.

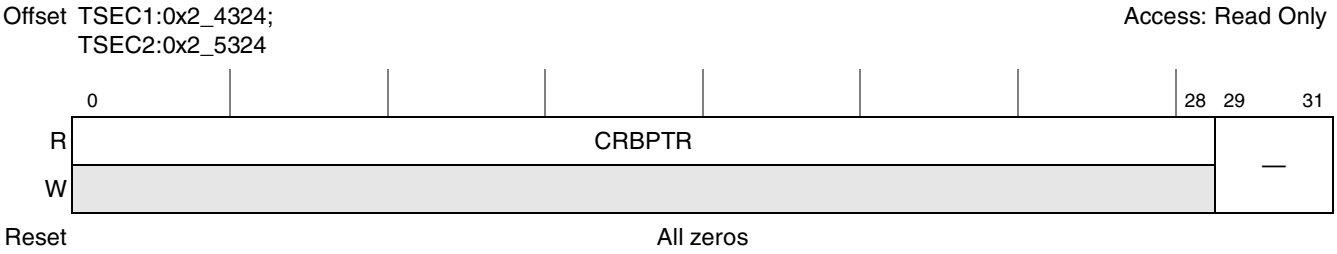


Figure 15-31. CRBPTR Register Definition

Table 15-29 describes the fields of the CRBPTR register.

Table 15-29. CRBPTR Field Descriptions

Bits	Name	Description
0–28	CRBPTR	CRBPTR is internally written by the DMA module. The value increments by one (causing the register value to increment by eight) each time a descriptor is read from memory.
29–31	—	Reserved

15.5.3.4.6 Maximum Receive Buffer Length Register (MRBLR)

MRBL is written by the user. It informs the TSEC how much space is in the receive buffer pointed to by the RxBD. Figure 15-32 shows the MRBLR.

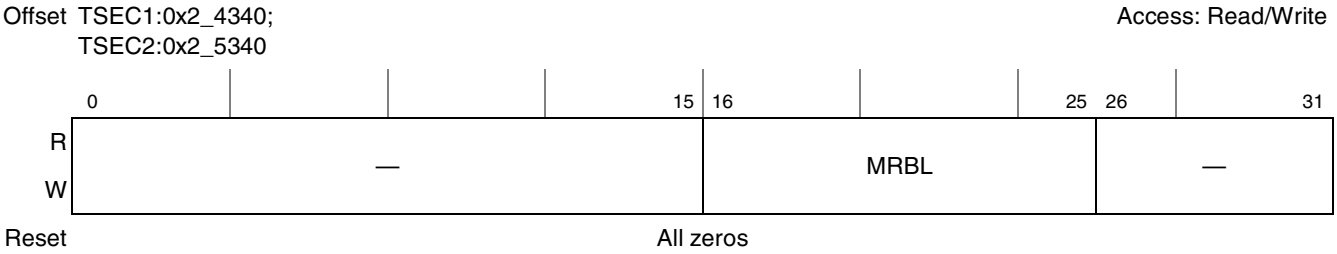


Figure 15-32. MRBL Register Definition

Table 15-30 describes the fields of the MRBL register.

Table 15-30. MRBLR Field Descriptions

Bits	Name	Description
0–15	—	To ensure that MRBL is a multiple of 64, these bits are reserved and must be cleared.
16–25	MRBL	Maximum receive buffer length. MRBL is the number of bytes that the TSEC receiver writes to the receive buffer. The MRBL register is written by the user with a multiple of 64 for all modes. TSEC 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.
26–31	—	To ensure that MRBL is a multiple of 64, these bits are reserved and must be cleared.

15.5.3.4.7 Receive Buffer Descriptor Pointer Register (RBPTR)

RBPTR, shown in Figure 15-33, contains the receive buffer descriptor address. This register takes on the value of RBASE when the RBASE register is written by software. Although not necessary in most applications, the user can modify this register when the transmitter has been gracefully stopped or halted, as indicated by TSTAT[THLT].

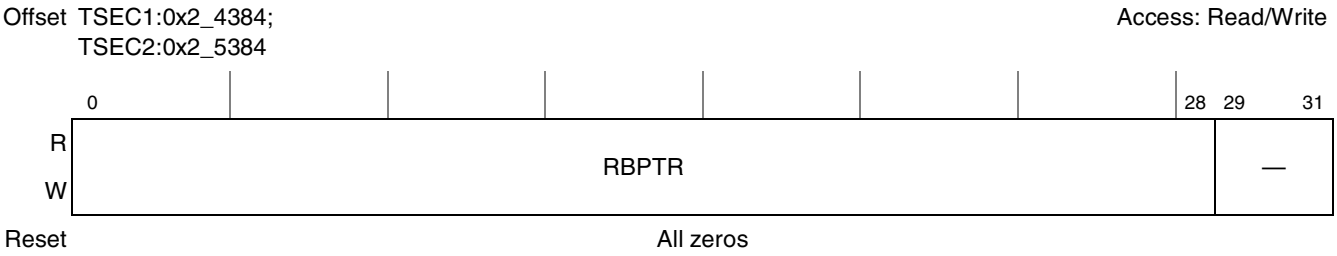


Figure 15-33. RBPTR Register Definition

Table 15-31 describes the fields of the RBPTR register.

Table 15-31. RBPTR Field Descriptions

Bits	Name	Description
0–28	RBPTR	RBPTR is internally written by the DMA module. The value increments (causing the register value to increment by eight) each time a descriptor is read from memory.
29–31	—	Reserved, should be cleared

15.5.3.4.8 Receive Descriptor Base Address Register (RBASE)

The RBASE register is written by the user with the RxBD base address and must be divisible by eight for the 8-byte buffer descriptors. Figure 15-34 shows the RBASE register.

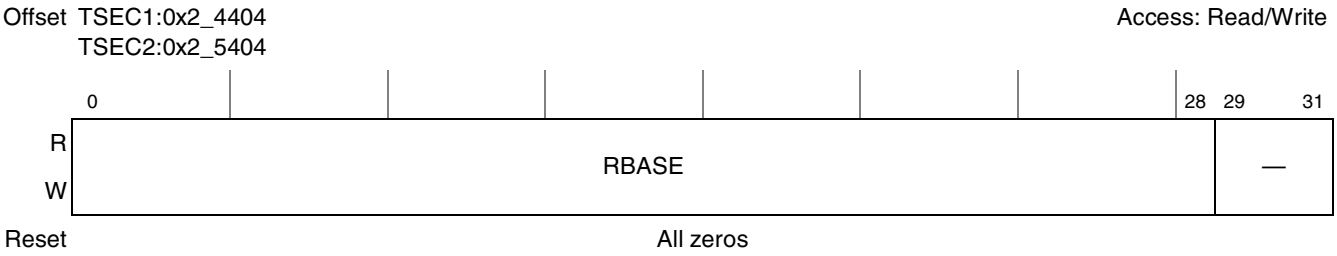


Figure 15-34. RBASE Register Definition

Table 15-32 describes the fields of the RBASE register.

Table 15-32. RBASE Field Descriptions

Bits	Name	Description
0–28	RBASE	Receive base. RBASE defines the starting location in the memory map for the TSEC RxBD.
29–31	—	Reserved. These bits must be zeros to cause the value of RBASE to be a multiple of eight.

15.5.3.5 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.

15.5.3.5.1 Configuring the MAC

MAC configuration registers 1 and 2 provide a way to configure 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.
- Varying pad/CRC combinations—Three pad/CRC combinations are provided to handle a variety of system requirements. The most simple are frames that already have a valid FCS field. In this case, the CRC is checked and reported via the transmit statistics vector (TSV[51:0]). 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.

15.5.3.5.2 Controlling CSMA/CD

The half-duplex register (HAFDUP) allows control over the carrier-sense multiple access/collision detection (CSMA/CD) logic of the TSEC. Half-duplex is only supported for 10 Mbps 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 TSEC 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.

15.5.3.5.3 Handling Packet Collisions

While transmitting a packet in half-duplex mode, the TSEC 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 TSEC 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 via 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 must flush the frame and move to the next one in line. If the system requests to send a packet while the TSEC is deferring to a carrier, the TSEC 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 TSEC 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, TSEC backs-off either 0 or 1 slot times. After the fifth collision, TSEC 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 via 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.

15.5.3.5.4 Controlling Packet Flow

Packet flow can be dealt with in a number of ways within TSEC. 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, the end occurs around the 56th byte of the frame data. Slot time in 1-Gbps 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 TSEC 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 TSEC 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, TSEC ensures the minimum 96-bit presence on the wire, then drops preamble and waits a back-off time depending on the value of the

configuration bit, back pressure no back-off (half-duplex) [BPNB]. These transmitting-preamble-for-back pressure collisions are not counted. If BPNB is set, the TSEC waits an inter-packet gap before resuming the transmission of preamble following the collision and does not defer. If cleared, the TSEC 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, BPNB must be set.

The TSEC drops carrier (cease transmitting preamble) periodically to avoid excessive defer conditions in other stations on the shared network. If, while applying back pressure, the TSEC is requested to send a packet, it stops sending preamble, and waits one IPG before sending the packet. BPNB applies for any collision that occurs during the sending of this packet. TSEC 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.

15.5.3.5.5 Controlling PHY Links

Control and status to and from the PHY is provided via the two-wire MII management interface described in IEEE 802.3u. The MII management registers (MII management configuration, command, address, control, status, and indicator registers) are used to exercise this interface between a host processor and one or more PHY devices (including the TBI). External PHYs may only be configured using the MII management interface of TSEC1 since the interface signals (EC_MDC and EC_MDIO) are only driven by TSEC1.

The TSEC MII's registers provide the ability to perform continuous read cycles, called a scan cycle. 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 Status].

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 TSEC. While enabled, the length of MII management frames are reduced from 64 clocks to 32 clocks. This effectively doubles the efficiency of the interface.

15.5.3.6 MAC Registers

Following are the MAC registers.

15.5.3.6.1 MAC Configuration Register 1 (MACCFG1)

Figure 15-35 shows MACCFG1.

Offset TSEC1:0x2_4500; TSEC2:0x2_5500 Access: Read/Write

	0	1														
R	Soft_Reset	—														
W																
Reset	All zeros															

	11	12	13	14	15
R	—	Reset Rx MC	Reset Tx MC	Reset Rx Fun	Reset Tx Fun
W					
Reset	All zeros				

	16	22	23
R	—		Loop Back
W			
Reset	All zeros		

	24	25	26	27	28	29	30	31
R	—		Rx_Flow	Tx_Flow	Sync'd Rx EN	Rx_EN	Sync'd Tx EN	Tx_EN
W								
Reset	All zeros							

Figure 15-35. MACCFG1 Register Definition

Table 15-33 describes the fields of the MACCFG1 register.

Table 15-33. MACCFG1 Field Descriptions

Bits	Name	Description
0	Soft_Reset	Soft reset. This bit is cleared by default. See Section 15.6.2.2, "Soft Reset and Reconfiguring Procedure," for more information on setting this bit. 0 Normal operation. 1 Place all modules within the MAC in reset.
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 MAC control block 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 PETMC transmit MAC control block 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 block in reset. This block performs the receive frame protocol.
15	Reset Tx Fun	Reset transmit function block. This bit is cleared by default. 0 Normal operation. 1 Place the transmit function block in reset. This block performs the frame transmission protocol.

Table 15-33. MACCFG1 Field Descriptions (continued)

Bits	Name	Description
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. 0 The receive MAC control ignores PAUSE flow control frames. 1 The receive MAC control detects and acts on PAUSE flow control frames.
27	Tx_Flow	Transmit flow. This bit is cleared by default. 0 The transmit MAC control may not send PAUSE flow control frames if requested by the system. 1 The transmit MAC control may send PAUSE flow control frames if requested by the system.
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. 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. 0 The MAC may not transmit frames from the system. 1 The MAC may transmit frames from the system.

15.5.3.6.2 MAC Configuration Register 2 (MACCFG2)

The MACCFG2 register is written by the user. Figure 15-36 shows the MACCFG2 register.

Offset TSEC1:0x2_4504; TSEC2:0x2_5504 — Access: Read/Write

	0															15
R	—															
W																
Reset	All zeros															

	16			19	20	21	22	23	24	25	26	27	28	29	29	31
R	Preamble Length				—		I/F Mode		—		Huge Frame	Length Check	—	PAD/ CRC	CRC EN	Full Duplex
W																
Reset	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-36. MACCFG2 Register Definition

Table 15-34 describes the fields of the MACCFG2 register.

Table 15-34. MACCFG2 Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–19	Preamble Length	Determines the length in bytes of the preamble field of the packet. Its default is 0x7. A preamble length of 0 is not supported. **Note:** TSEC does not function correctly when MACCFG2[PreambleLength] is programmed to certain values. The safe values for PreambleLength in GMII/TBI mode is PreambleLength > 2 and PreambleLength <= 15. In MII mode the safe values are PreambleLength > 0 and PreambleLength <= 7.
20–21	—	Reserved
22–23	I/F Mode	Determines the type of interface to which the MAC is connected. Its default is 00. 00 Reserved 01 Nibble mode (MII) 10 Byte mode (GMII/TBI) 11 Reserved
24–25	—	Reserved
26	Huge Frame	Huge frame enable. Cleared by default. 0 Limit the length of frames to the MAXIMUM FRAME LENGTH value. 1 Frames longer than the MAXIMUM FRAME LENGTH may be transmitted and received.
27	Length check	Length check. This bit is cleared by default. 0 No length field checking is performed. 1 The MAC checks the frame's length field to ensure it matches the actual data field length.
28	—	Reserved
29	PAD/CRC	Pad and append CRC. This bit is cleared by default. 0 Frames presented to the MAC have a valid length and contain a CRC. 1 The MAC pads all transmitted short frames and appends a CRC to every frame regardless of padding requirement.
30	CRC EN	CRC enable. If the configuration bit PAD/CRC or the per-packet PAD/CRC bit is set, CRC EN is ignored. This bit is cleared by default. 0 Frames presented to the MAC have a valid length and contain a valid CRC. 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.
31	FULL DUPLEX	Full duplex configure. This bit is cleared by default. 0 The MAC operates in half-duplex mode. 1 The MAC operates in full-duplex mode.

15.5.3.6.3 Inter-Packet Gap/Inter-Frame Gap Register (IPGIFG)

The IPGIFG register is written by the user. Figure 15-37 shows the IPGIFG register.

Offset TSEC1:0x2_4508; TSEC2:0x2_5508 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	—	Non-Back-to Back Inter-Packet-Gap, Part 1							—	Non-Back-to Back Inter-Packet-Gap, Part 2						
Reset	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	Minimum IFG Enforcement								—	Back-to Back Inter-Packet-Gap						
Reset	0	1	0	1	0	0	0	0	0	1	1	0	0	0	0	0

Figure 15-37. IPGIFG Register Definition

Table 15-35 describes the fields of the IPGIFG register.

Table 15-35. IPGIFG Field Descriptions

Bits	Name	Description
0	—	Reserved
1–7	Non-Back-to-Back Inter-Packet-Gap, Part 1	Optional carrier sense window referenced in IEEE 802.3/4.2.3.2.1 as carrier deference. If carrier is detected during the timing of IPGR1, the MAC defers to the carrier. If, however, the 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, which follows the two-thirds/one-third guideline.
8	—	Reserved
9–15	Non-Back-to-Back Inter-Packet-Gap, Part 2	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	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 represents half of the nominal minimum IFG which is 160 bits.
24	—	Reserved
25–31	Back-to-Back Inter-Packet-Gap	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.

15.5.3.6.4 Half-Duplex Register (HAFDUP)

The HAFDUP register is written by the user. Figure 15-38 shows the HAFDUP register.

Offset TSEC1:0x2_450C; TSEC2:0x2_550C — Access: Read/Write

	0				7	8			11	12	13	14	15
R W	—					Alternate BEB Truncation				Alt BEB	BP No BackOff	No BackOff	Excess Defer
Reset	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1

	16			19	20			25	26			31
R W	Retransmission Maximum				—				Collision Window			
Reset	1	1	1	1	0	0	0	0	0	0	1	1	0	1	1	1

Figure 15-38. Half-Duplex Register Definition

Table 15-36 describes the fields of the HAFDUP register.

Table 15-36. HAFDUP Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8–11	Alternate BEB Truncation	This field is used while alternate binary exponential back-off 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 back-off. 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 back-off 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 back-off time.
13	BP No BackOff	Back pressure no back-off. 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 back-off. 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.

Table 15-36. HAFDUP Field Descriptions (continued)

Bits	Name	Description
20–25	—	Reserved
26–31	Collision Window	This is a programmable field representing the slot time or collision window during which collisions occur in properly configured networks. Because the collision window starts at the beginning of transmission, the preamble and SFD are included. Its default of 0x37 corresponds to the count of frame bytes at the end of the window.

15.5.3.6.5 Maximum Frame Length Register (MAXFRM)

The MAXFRM register is written by the user. Figure 15-39 shows the MAXFRM register.

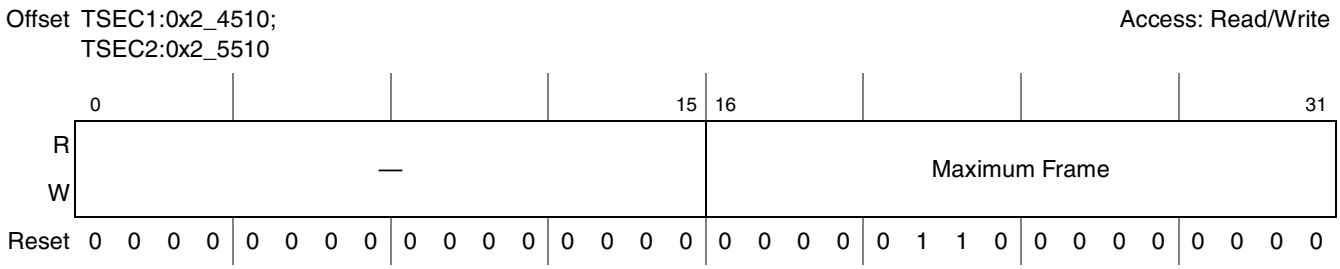


Figure 15-39. Maximum Frame Length Register Definition

Table 15-37 describes the fields of the MAXFRM register.

Table 15-37. MAXFRM Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	Maximum Frame	It sets the maximum frame size in both the transmit and receive directions. (Refer to MACCFG2[Huge Frame].)

15.5.3.6.6 MII Management Configuration Register (MIIMCFG)

The MIIMCFG register is written by the user. Figure 15-40 shows the MIIMCFG register.

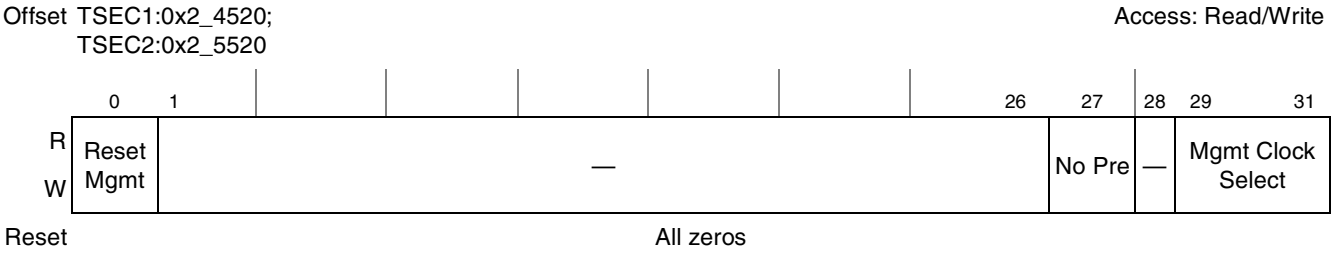


Figure 15-40. MII Management Configuration Register Definition

Table 15-38 describes the fields of the MIIMCFG register.

Table 15-38. MIIMCFG Field Descriptions

Bits	Name	Description
0	Reset Mgmt	Reset management. This bit is cleared by default. 0 Allow the MII Mgmt to perform management read/write cycles if requested. 1 Reset the MII Mgmt.
1–26	—	Reserved
27	No Pre	Preamble suppress. This bit is cleared by default. 0 The MII Mgmt performs management read/write cycles with 32 clocks of preamble. 1 The MII Mgmt suppresses preamble generation and reduces the management cycle from 64 clocks to 32 clocks. This is in accordance with IEEE 802.3/22.2.4.4.2.
28	—	Reserved
29–31	Mgmt Clock Select	This field determines the clock frequency of the Mgmt Clock (EC_MDC). Its default value is 000. The source clock frequency is equal to the TSEC internal clock divided by eight. Note that TSEC internal clock frequency is affected by the value programmed in register OCCR[TSECnCM]. 000 Source clock divided by 4 001 Source clock divided by 4 010 Source clock divided by 6 011 Source clock divided by 8 100 Source clock divided by 10 101 Source clock divided by 14 110 Source clock divided by 20 111 Source clock divided by 28

15.5.3.6.7 MII Management Command Register (MIIMCOM)

The MIIMCOM register is written by the user. Figure 15-41 shows the MIIMCOM register.

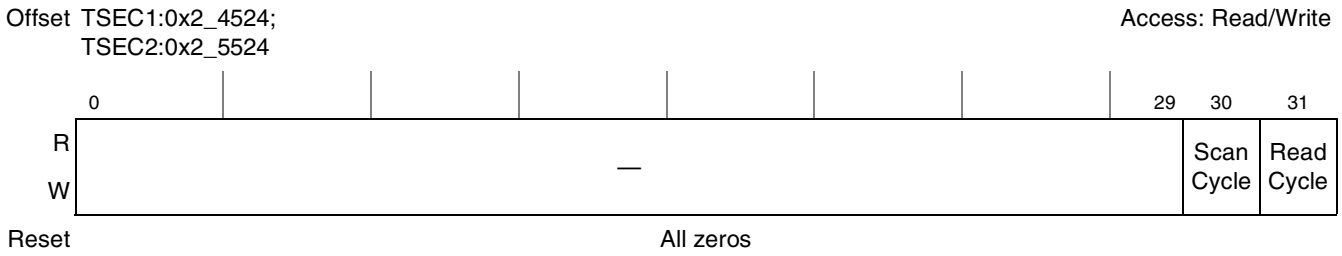


Figure 15-41. MIIMCOM Register Definition

Table 15-39 describes the fields of the MIIMCOM register.

Table 15-39. MIIMCOM Field Descriptions

Bits	Name	Description
0–29	—	Reserved

Table 15-39. MIIMCOM Field Descriptions (continued)

Bits	Name	Description
30	Scan Cycle	Scan cycle. This bit is cleared by default. 0 Normal operation. 1 The MII management continuously performs read cycles. This is useful for polling a PHY register, for example, monitoring link fails. Data is returned in register MIIMSTAT[PHY Status]. IEVENT[MMRD] is set once a read has been completed.
31	Read Cycle	Read cycle. This bit is cleared by default but is not self-clearing once set. 0 Normal operation. 1 The MII management performs a single read cycle upon the transition of this bit from 0 to 1 using the PHY address (at MIIMADD[PHY Address]) and the register address (at MIIMADD[Register Address]). The 0 to 1 transition of this bit also causes the MIIMIND[BUSY] bit to be set. The read is complete when MIIMIND[BUSY] clears. Data is returned in register MIIMSTAT[PHY Status]. IEVENT[MMRD] is also set once the read has been completed.

15.5.3.6.8 MII Management Address Register (MIIMADD)

The MIIMADD register is written by the user. Figure 15-42 shows the MIIMADD register.

Offset TSEC1:0x2_4528; TSEC2:0x2_5528 — Access: Read/Write

	0–18	19–23	24–26	27–31
R W	—	PHY Address	—	Register Address
Reset	All zeros			

Figure 15-42. MIIMADD Register Definition

Table 15-40 describes the fields of the MIIMADD register.

Table 15-40. MIIMADD Field Descriptions

Bits	Name	Description
0–18	—	Reserved
19–23	PHY Address	PHY address field of management cycles. Up to 31 PHYs can be addressed (0 is reserved).
24–26	—	Reserved
27–31	Register Address	Represents the 5-bit register address field of management cycles. Up to 32 registers can be accessed. Its default value is 0x00.

15.5.3.6.9 MII Management Control Register (MIIMCON)

The MIIMCON register is written by the user. Figure 15-43 shows the MIIMCON register.

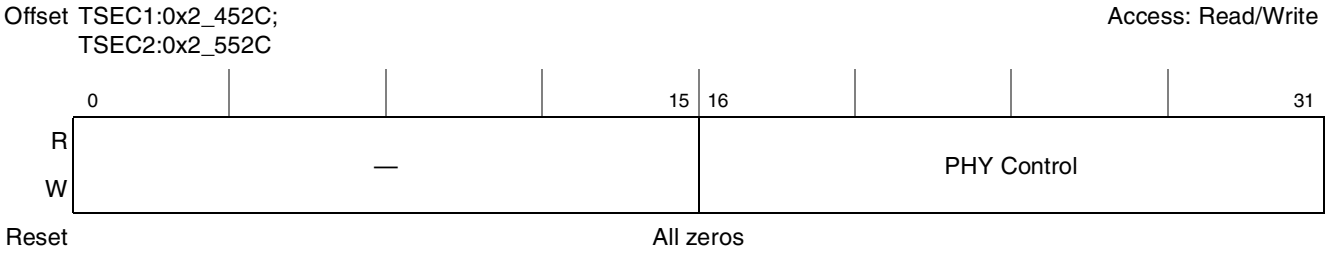


Figure 15-43. MII Management Control Register Definition

Table 15-41 describes the fields of the MIIMCON register.

Table 15-41. 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. IEVENT[MMWR] is set once the write has been completed.

15.5.3.6.10 MII Management Status Register (MIIMSTAT)

The MIIMSTAT, shown in Figure 15-44, is read-only by the user.

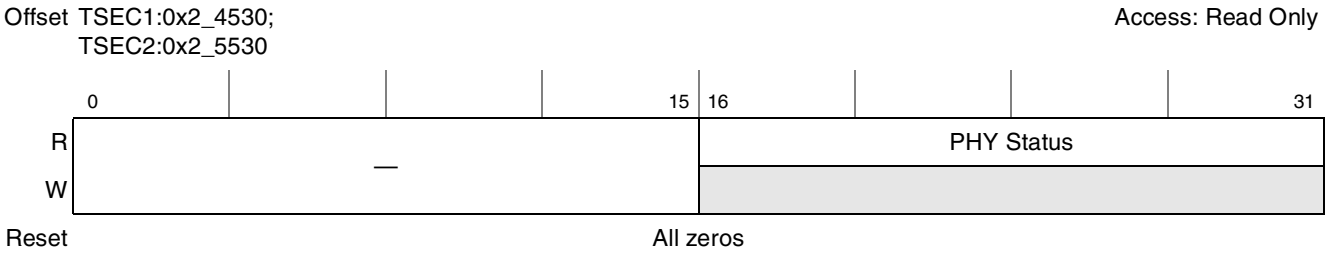


Figure 15-44. MIIMSTAT Register Definition

Table 15-42 describes the fields of the MIIMSTAT register.

Table 15-42. 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.

15.5.3.6.11 MII Management Indicator Register (MIIMIND)

The MIIMIND register is read-only by the user. Figure 15-45 shows the MIIMIND register.

Offset TSEC1:0x2_4534; TSEC2:0x2_5534 Access: Read Only

	0							28	29	30	31
R	—								Not Valid	Scan	Busy
W											
Reset	All zeros										

Figure 15-45. MII Management Indicator Register Definition

Table 15-43 describes the fields of the MIIMIND register.

Table 15-43. 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. IEVENT[MMRD] or IEVENT[MMWR] is set once the read or write, respectively, has been completed.

15.5.3.6.12 Interface Status Register (IFSTAT)

The IFSTAT register is written by the user. Figure 15-46 shows the IFSTAT register.

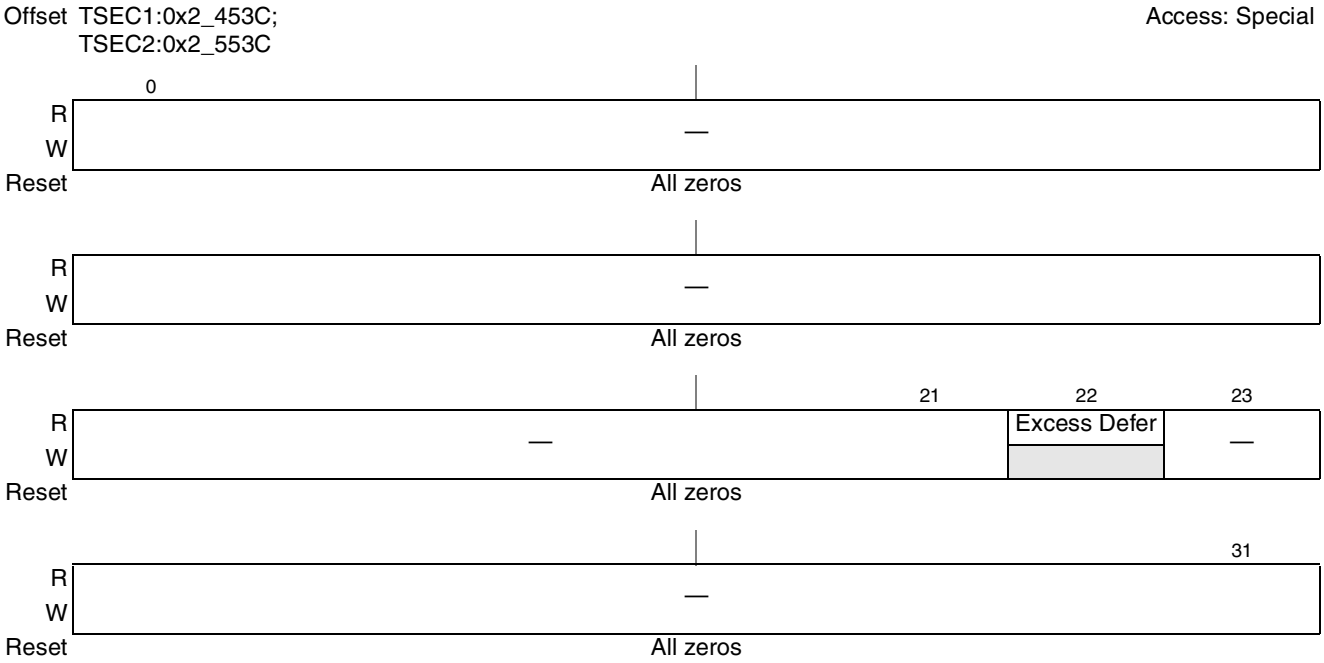


Figure 15-46. IFSTAT Register Definition

Table 15-44 describes the fields of the IFSTAT register.

Table 15-44. 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

15.5.3.6.13 Station Address Register Part 1 (MACSTNADDR1)

The MACSTNADDR1 register is written by the user. Figure 15-47 shows the MACSTNADDR1 register. The value of the station address written by the user into MACSTNADDR1 and MACSTNADDR2 is byte-reversed from how it would appear in the DA field of a frame in memory. For example, for a station address of 0x12345678ABCD, perform a write to MACSTNADDR1 of 0xCDAB7856, and to MACSTNADDR2 of 0x34120000. When the user reads MACSTNADDR1, 0xCDAB7856 is returned. A read of MACSTNADDR2 returns a value of 0x34120000. Note, the I/G and U/L bits of the frame's DA field is located at the LSBs of the 1st octet stored in MACSTNADDR2, where the I/G bit is bit 15, and the U/L bit is bit 14.

Offset TSEC1:0x2_4540; TSEC2:0x2_5540 Access: Read/Write

	0–7	8–15	16–23	24–31
R W	Station Address,6th Octet	Station Address,5th Octet	Station Address,4th Octet	Station Address,3rd Octet
Reset	All zeros			

Figure 15-47. MACSTNADDR1 Register Definition

Table 15-45 describes the fields of the MACSTNADDR1 register.

Table 15-45. MACSTNADDR1 Field Descriptions

Bits	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 0x00.
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 0x00.
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 0x00.
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 0x00.

15.5.3.6.14 Station Address Register Part 2 (MACSTNADDR2)

The MACSTNADDR2 register is written by the user. Figure 15-48 shows the MACSTNADDR2 register. Note, the I/G and U/L bits of the frame's DA field is located at the LSBs of the 1st octet stored in MACSTNADDR2, where the I/G bit is bit 15, and the U/L bit is bit 14.

Offset TSEC1:0x2_4544; TSEC2:0x2_5544 Access: Read/Write

	0–7	8–15	16–31
R W	Station Address, 2nd Octet	Station Address, 1st Octet	—
Reset	All zeros		

Figure 15-48. MACSTNADDR2 Register Definition

Table 15-46 describes the fields of the MACSTNADDR2 register.

Table 15-46. MACSTNADDR2 Field Descriptions

Bits	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 0x00.

Table 15-46. MACSTNADDR2 Field Descriptions (continued)

Bits	Name	Description
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 0x00.
16–31	—	Reserved

15.5.3.7 MIB Registers

The TSEC MSTAT module has 37 separate statistics counters, which count or accumulate statistical events that occur as packets are transmitted and received. These counters support RMON MIB group 1, RMON MIB group 2, RMON MIB group 3, RMON MIB group 9, RMON MIB 2, and the 802.3 Ethernet MIB.

The detection of one or more of these statistical events triggers the MSTAT module to update its statistics counters. These counters are stored in internal data registers. The user may access the internal data registers at any time. An interrupt can be generated upon any one counter's rollover condition via a carry interrupt output from the MSTAT. Each counter's rollover condition can be discreetly 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 assertion of an external module input signal. Figure 15-49 shows the TR64 register.

15.5.3.7.1 Transmit and Receive 64-Byte Frame Counter Register (TR64)

Figure 15-49 shows the TR64 register.

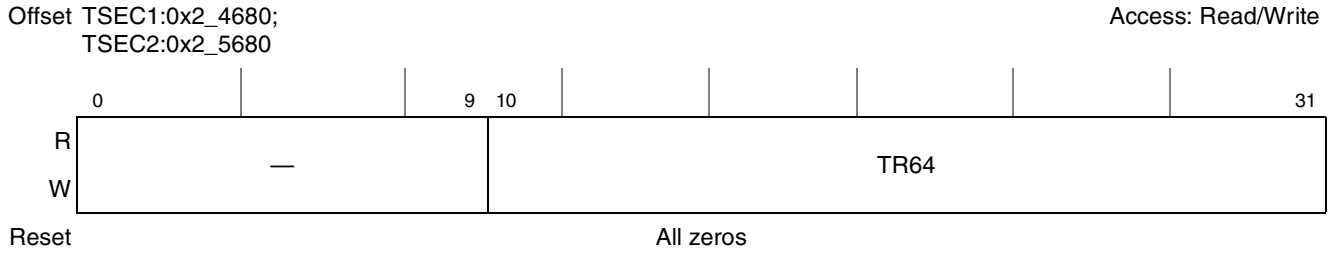


Figure 15-49. Transmit and Receive 64-Byte Frame Register Definition

Table 15-47 describes the fields of the TR64 register.

Table 15-47. 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)

15.5.3.7.2 Transmit and Receive 65- to 127-Byte Frame Counter Register (TR127)

Figure 15-50 shows the TR127 register.

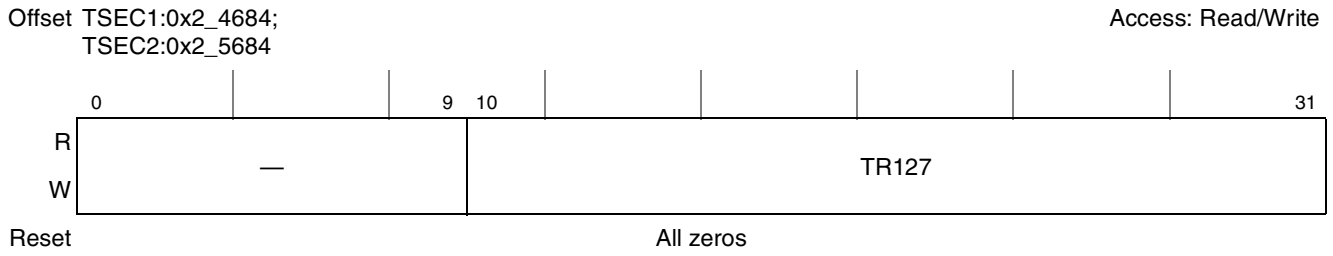


Figure 15-50. Transmit and Receive 65- to 127-Byte Frame Register Definition

Table 15-48 describes the fields of the TR127 register.

Table 15-48. TR127 Field Descriptions

Bits	Name	Description
0–9	—	Reserved
10–31	TR127	Transmit and receive 65- to 127-byte frame counter—Increment for each good or bad frame transmitted and received which is 65 to 127 bytes in length, inclusive (excluding preamble and SFD but including FCS bytes)

15.5.3.7.3 Transmit and Receive 128- to 255-Byte Frame Counter Register (TR255)

Figure 15-51 shows the TR255 register.

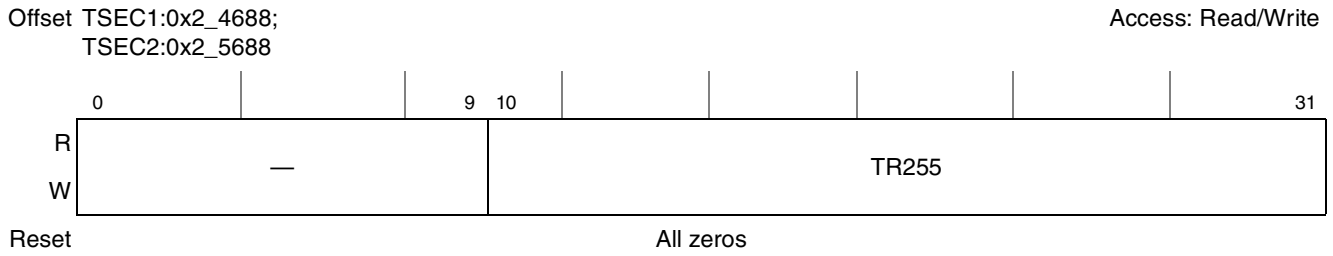


Figure 15-51. Transmit and Received 128- to 255-Byte Frame Register Definition

Table 15-49 describes the fields of the TR255 register.

Table 15-49. 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 to 255 bytes in length, inclusive (excluding preamble and SFD but including FCS bytes)

15.5.3.7.4 Transmit and Receive 256- to 511-Byte Frame Counter Register (TR511)

Figure 15-52 shows the TR511 register.

Offset TSEC1:0x2_468C; TSEC2:0x2_568C — Access: Read/Write

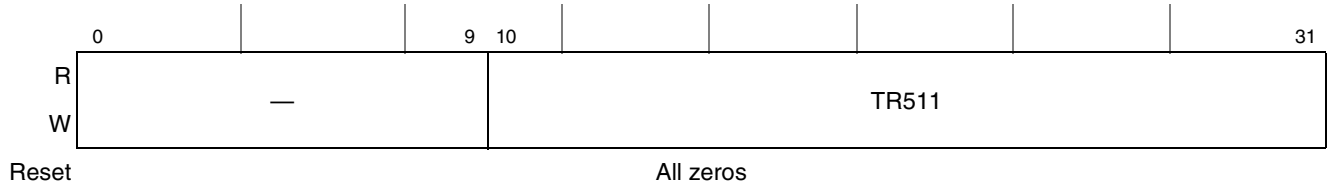


Figure 15-52. Transmit and Received 256- to 511-Byte Frame Register Definition

Table 15-50 describes the fields of the TR511 register.

Table 15-50. 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 to 511 bytes in length, inclusive (excluding preamble and SFD but including FCS bytes)

15.5.3.7.5 Transmit and Receive 512- to 1023-Byte Frame Counter Register (TR1K)

Figure 15-53 shows the TR1K register.

Offset TSEC1:0x2_4690; TSEC2:0x2_5690 — Access: Read/Write

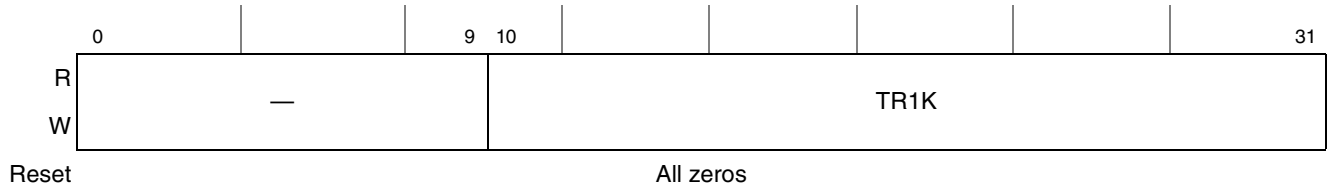


Figure 15-53. Transmit and Received 512- to 1023-Byte Frame Register Definition

Table 15-51 describes the fields of the TR1K register.

Table 15-51. 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 to 1023 bytes in length, inclusive (excluding preamble and SFD but including FCS bytes)

15.5.3.7.6 Transmit and Receive 1024- to 1518-Byte Frame Counter Register (TRMAX)

Figure 15-54 shows the TRMAX register.

Offset TSEC1:0x2_4694; TSEC2:0x2_5694 Access: Read/Write

	0	9	10	31
R	—		TRMAX	
W				
Reset	All zeros			

Figure 15-54. Transmit and Received 1024- to 1518-Byte Frame Register Definition

Table 15-52 describes the fields of the TRMAX register.

Table 15-52. 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 to 1518 bytes in length, inclusive (excluding preamble and SFD but including FCS bytes)

15.5.3.7.7 Transmit and Receive 1519- to 1522-Byte VLAN Frame Counter Register (TRMGV)

Figure 15-56 shows the TRMGV register.

Offset TSEC1:0x2_4698; TSEC2:0x2_5698 Access: Read/Write

	0	9	10	31
R	—		TRMGV	
W				
Reset	All zeros			

Figure 15-55. Transmit and Received 1519- to 1522-Byte VLAN Frame Register Definition

Table 15-53 describes the fields of the TRMGV register.

Table 15-53. 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 to 1522 bytes in length, inclusive (excluding preamble and SFD but including FCS bytes)

15.5.3.7.8 Receive Byte Counter Register (RBYT)

Figure 15-57 shows the RBYT register.

Offset TSEC1:0x2_469C; TSEC2:0x2_569C — Access: Read/Write

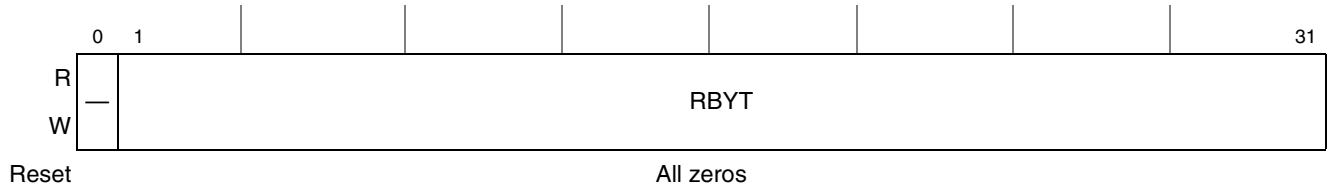


Figure 15-56. Receive Byte Counter Register Definition

Table 15-54 describes the fields of the RBYT register.

Table 15-54. RBYT Field Descriptions

Bits	Name	Description
0	—	Reserved
1–31	RBYT	Receive byte counter. Increments by the byte count of frames received, including those in bad packets, excluding preamble and SFD but including FCS bytes.

15.5.3.7.9 Receive Packet Counter Register (RPKT)

Figure 15-58 shows the RPKT register.

Offset TSEC1:0x2_46A0; TSEC2:0x2_56A0 — Access: Read/Write

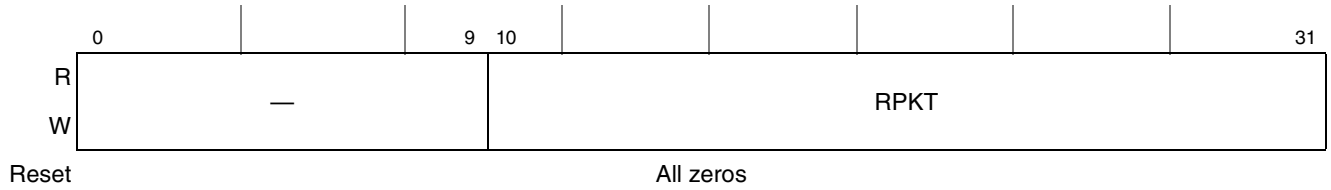


Figure 15-57. Receive Packet Counter Register Definition

Table 15-55 describes the fields of the RPKT register.

Table 15-55. 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).

15.5.3.7.10 Receive FCS Error Counter Register (RFCS)

Figure 15-59 shows the RFCS register.

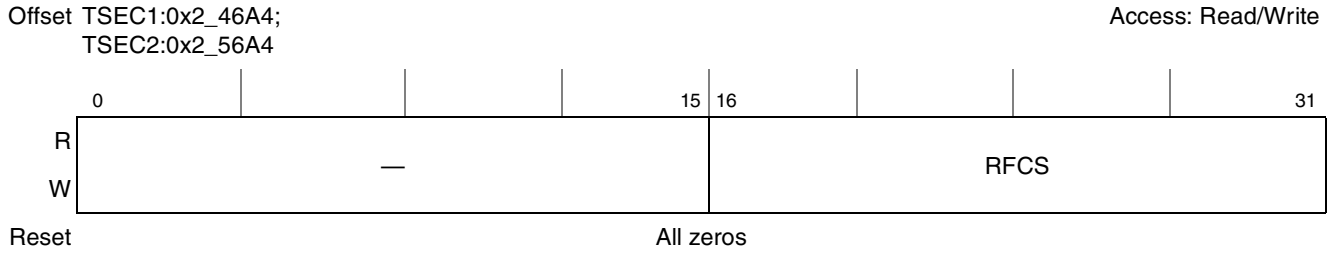


Figure 15-58. Receive FCS Error Counter Register Definition

Table 15-56 describes the fields of the RFCS register.

Table 15-56. RFCS Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	RFCS	Receive FCS error counter. Increments for each frame received that has an integral 64 to 1518 length and contains a frame check sequence error.

15.5.3.7.11 Receive Multicast Packet Counter Register (RMCA)

Figure 15-60 shows the RMCA register.

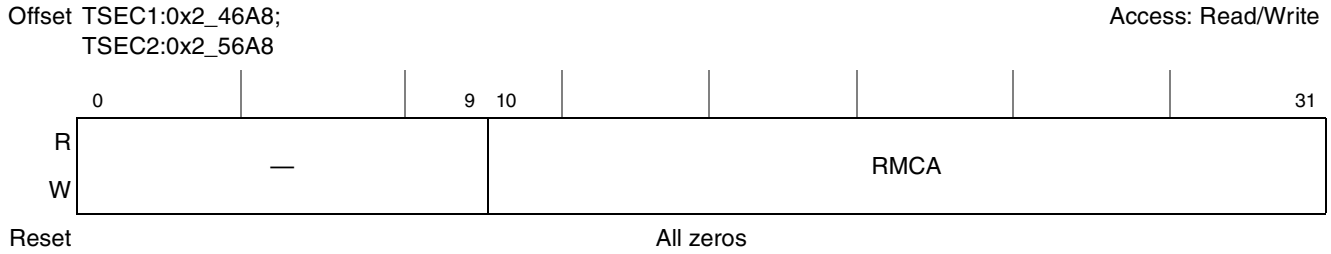


Figure 15-59. Receive Multicast Packet Counter Register Definition

Table 15-57 describes the fields of the RMCA register.

Table 15-57. RMCA Field Descriptions

Bits	Name	Description
0–9	—	Reserved
10–31	RMCA	Receive multicast packet counter. Increments for each multicast good frame of lengths 64 to 1518 (non VLAN) or 1522 (VLAN), excluding broadcast frames. This count does not include range/length errors.

15.5.3.7.12 Receive Broadcast Packet Counter Register (RBCA)

Figure 15-60 shows the RBCA register.

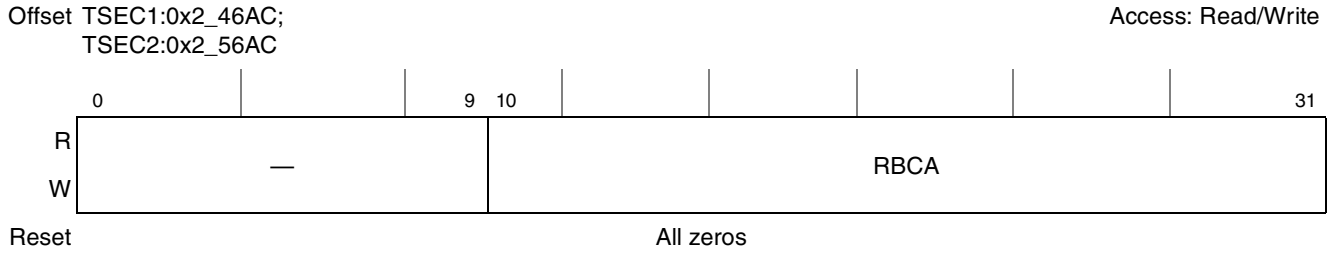


Figure 15-60. Receive Broadcast Packet Counter Register Definition

Table 15-58 describes the fields of the RBCA register.

Table 15-58. RBCA Field Descriptions

Bits	Name	Description
0–9	—	Reserved
10–31	RBCA	Receive broadcast packet counter. Increments for each broadcast good frame of lengths 64 to 1518 (non VLAN) or 1522 (VLAN), excluding multicast frames. Does not include range/length errors.

15.5.3.7.13 Receive Control Frame Packet Counter Register (RXCF)

Figure 15-62 shows the RXCF register.

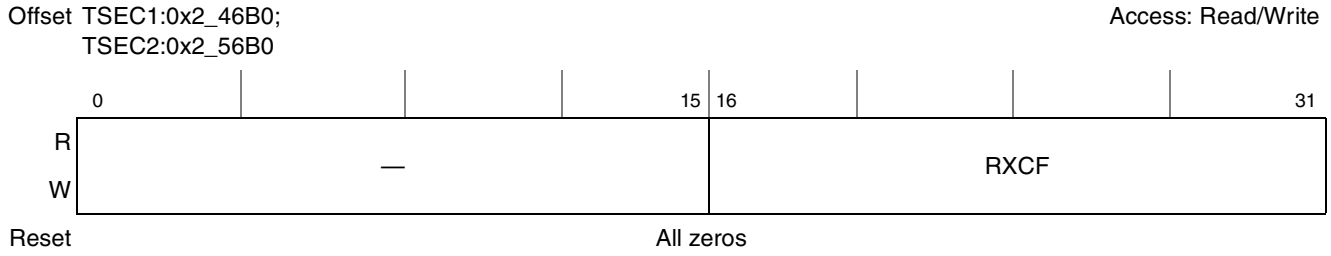


Figure 15-61. Receive Control Frame Packet Counter Register Definition

Table 15-59 describes the fields of the RXCF register.

Table 15-59. RXCF Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	RXCF	Receive control frame packet counter. Increments for each MAC control frame received (both pause control frames and control frames unsupported by IEEE).

15.5.3.7.14 Receive Pause Frame Packet Counter Register (RXPF)

Figure 15-63 shows the RXPF register.

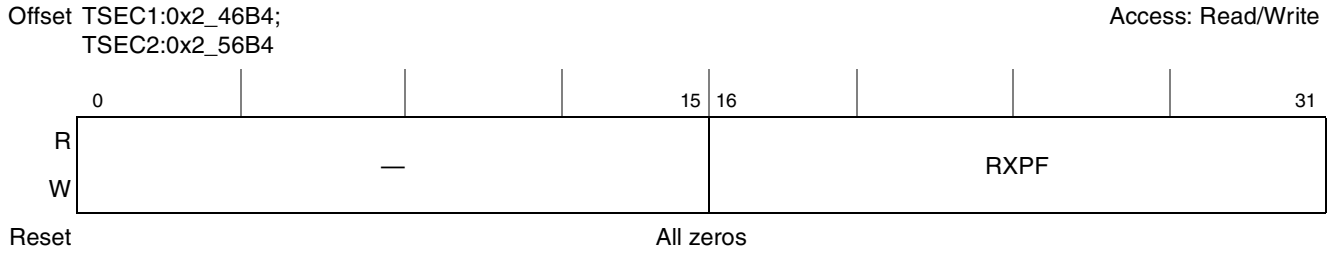


Figure 15-62. Receive Pause Frame Packet Counter Register Definition

Table 15-60 describes the fields of the RXPF register.

Table 15-60. RXPF Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	RXPF	Receive PAUSE frame packet counter. Increments each time a valid PAUSE MAC control frame is received.

15.5.3.7.15 Receive Unknown Opcode Packet Counter Register (RXUO)

Figure 15-64 shows the RXUO register.

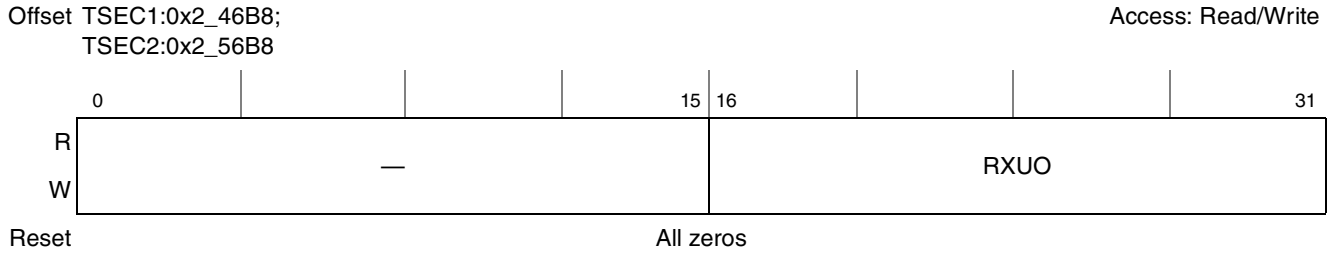


Figure 15-63. Receive Unknown Opcode Packet Counter Register Definition

Table 15-61 describes the fields of the RXUO register.

Table 15-61. 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 a PAUSE.

15.5.3.7.16 Receive Alignment Error Counter Register (RALN)

Figure 15-65 shows the RALN register.

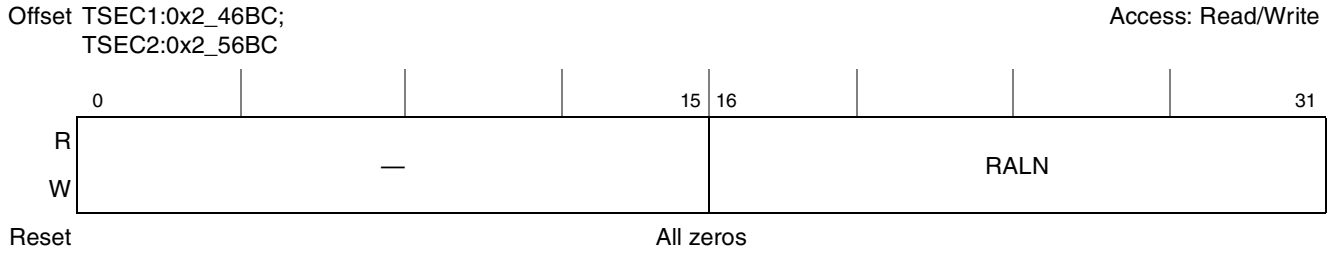


Figure 15-64. Receive Alignment Error Counter Register Definition

Table 15-62 describes the fields of the RALN register.

Table 15-62. 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.

15.5.3.7.17 Receive Frame Length Error Counter Register (RFLR)

Figure 15-66 shows the RFLR register.

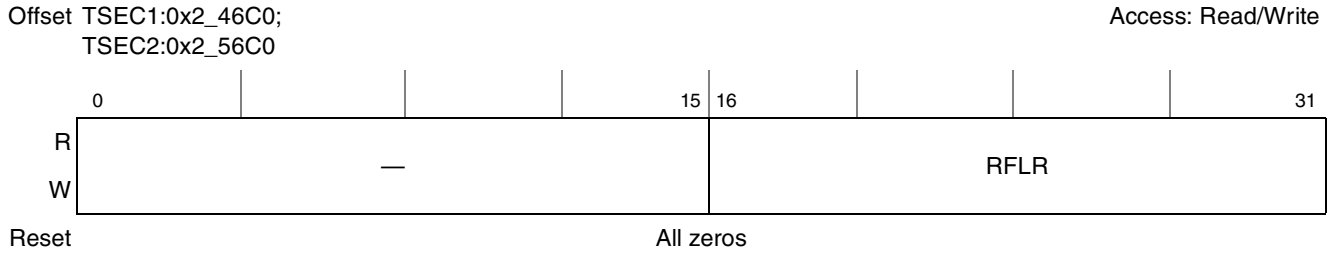


Figure 15-65. Receive Frame Length Error Counter Register Definition

Table 15-63 describes the fields of the RFLR register.

Table 15-63. 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.

15.5.3.7.18 Receive Code Error Counter Register (RCDE)

Figure 15-67 shows the RCDE register.

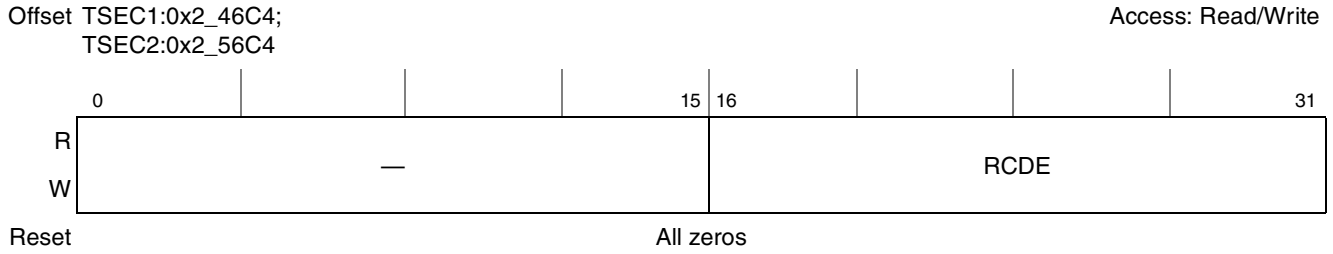


Figure 15-66. Receive Code Error Counter Register Definition

Table 15-64 describes the fields of the RCDE register.

Table 15-64. 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.

15.5.3.7.19 Receive Carrier Sense Error Counter Register (RCSE)

Figure 15-68 shows the RCSE register.

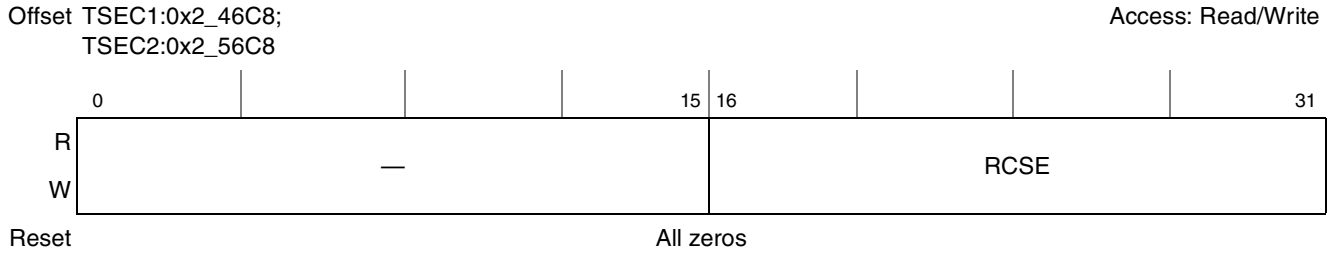


Figure 15-67. Receive Carrier Sense Error Counter Register Definition

Table 15-65 describes the fields of the RCSE register.

Table 15-65. RCSE Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	RCSE	Receive false carrier counter. Increments each time a false carrier is detected during idle, as defined by a 1 on TSEC*n*_RX_ER and an 0xE on TSEC*n*_RXD. The event is reported along with the statistics generated on the next received frame. Only one false carrier condition can be detected and logged between frames.

15.5.3.7.20 Receive Undersize Packet Counter Register (RUND)

Figure 15-69 shows the RUND register.

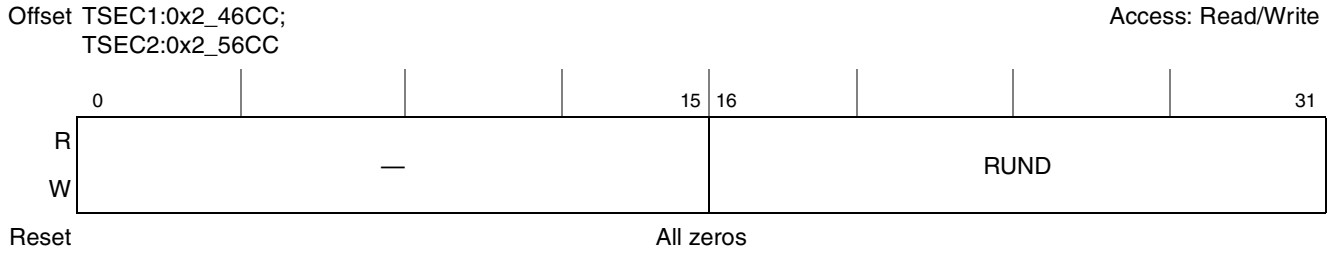


Figure 15-68. Receive Undersize Packet Counter Register Definition

Table 15-66 describes the fields of the RUND register.

Table 15-66. RUND Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	RUND	Receive undersize packet counter. Increments each time a frame is received that is less than 64 bytes and contains a valid FCS and were otherwise well formed. This count does not include range length errors.

15.5.3.7.21 Receive Oversize Packet Counter Register (ROVR)

Figure 15-70 shows the ROVR register.

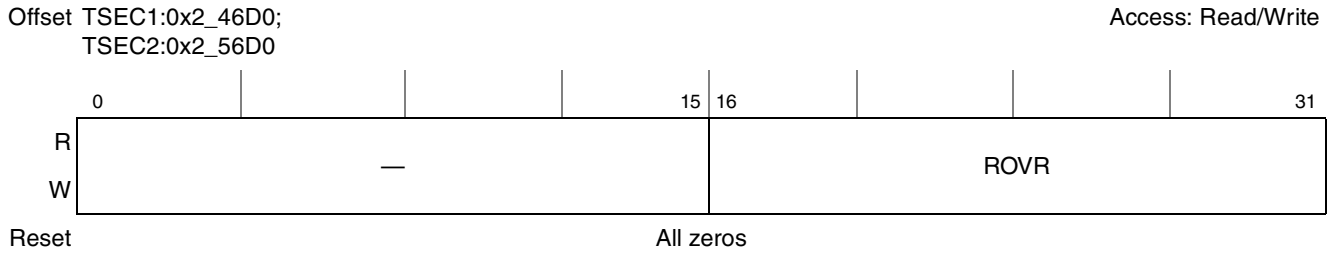


Figure 15-69. Receive Oversize Packet Counter Register Definition

Table 15-67 describes the fields of the ROVR register.

Table 15-67. ROVR Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	ROVR	Receive oversize packet counter. Increments each time a frame is received that exceeds 1518 (non VLAN) or 1522 (VLAN) and contains a valid FCS and was otherwise well formed. This count does not include range length errors.

15.5.3.7.22 Receive Fragments Counter Register (RFRG)

Figure 15-71 shows the RFRG register.

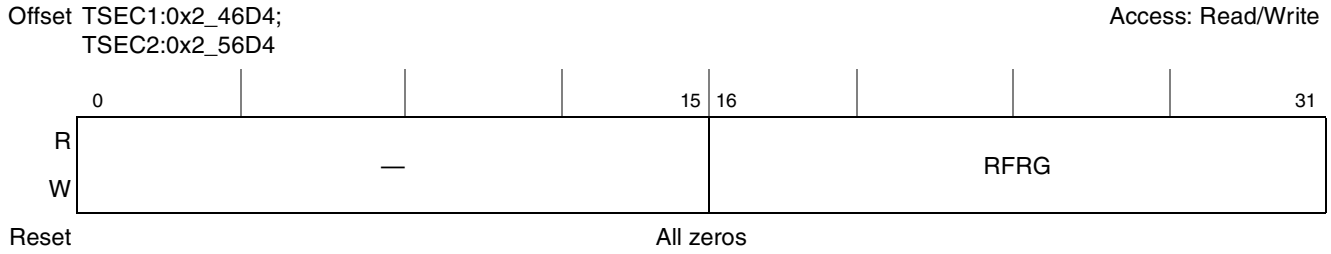


Figure 15-70. Receive Fragments Counter Register Definition

Table 15-68 describes the fields of the RFRG register.

Table 15-68. RFRG Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	RFRG	Receive fragments counter. Increments for each frame received that is less than 64 bytes and contains an invalid FCS. This includes integral and non-integral lengths.

15.5.3.7.23 Receive Jabber Counter Register (RJBR)

Figure 15-72 shows the RJBR register.

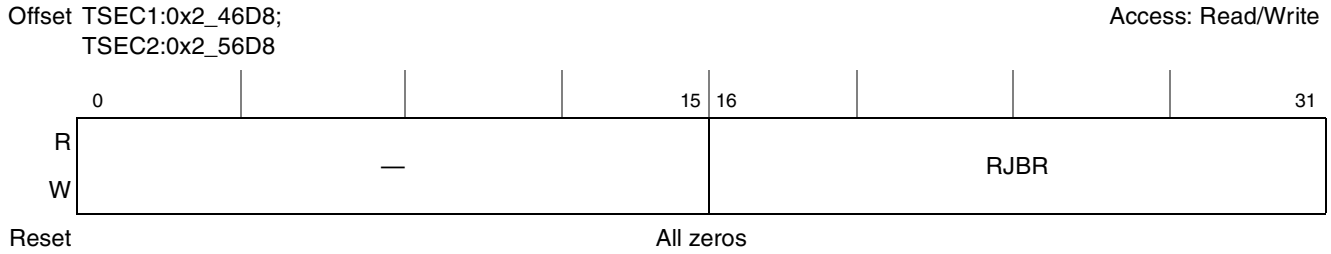


Figure 15-71. Receive Jabber Counter Register Definition

Table 15-69 describes the fields of the RJBR register.

Table 15-69. RJBR Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	RJBR	Receive jabber counter. Increments for frames received that exceed 1518 (non VLAN) or 1522 (VLAN) bytes and contain an invalid FCS. This includes alignment errors.

15.5.3.7.24 Receive Dropped Packet Counter Register (RDRP)

Figure 15-73 shows the RDRP register.

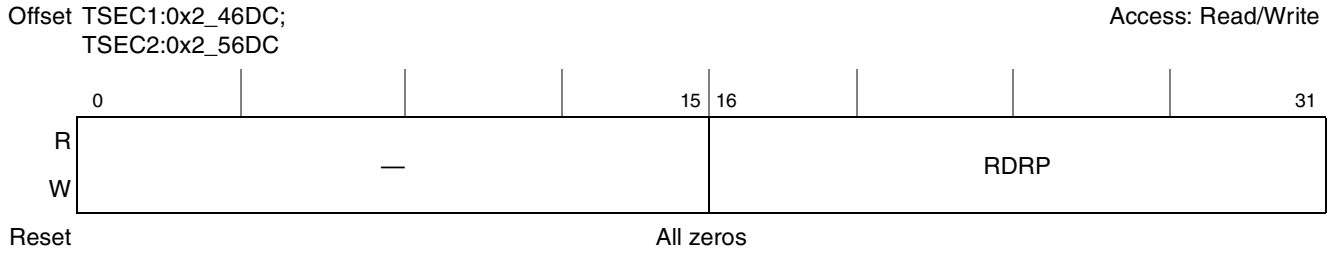


Figure 15-72. Receive Dropped Packet Counter Register Definition

Table 15-70 describes the fields of the RDRP register.

Table 15-70. 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.

15.5.3.7.25 Transmit Byte Counter Register (TBYT)

Figure 15-74 shows the TBYT register.

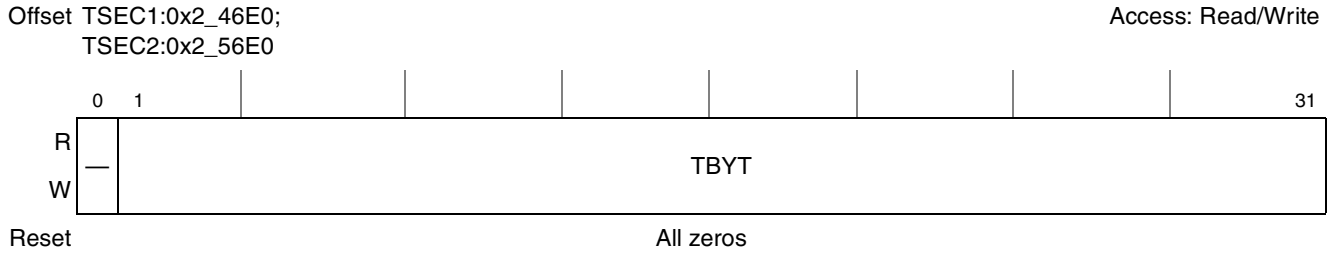


Figure 15-73. Transmit Byte Counter Register Definition

Table 15-71 describes the fields of the TBYT register.

Table 15-71. TBYT Field Descriptions

Bits	Name	Description
0	—	Reserved
1–31	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. This counter does not count if the frame is truncated.

15.5.3.7.26 Transmit Packet Counter Register (TPKT)

Figure 15-75 shows the TPKT register.

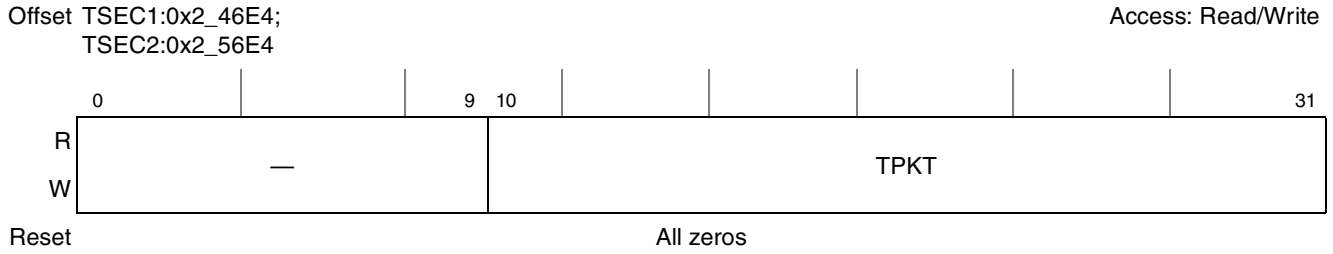


Figure 15-74. Transmit Packet Counter Register Definition

Table 15-72 describes the fields of the TPKT register.

Table 15-72. 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).

15.5.3.7.27 Transmit Multicast Packet Counter Register (TMCA)

Figure 15-76 shows the TMCA register.

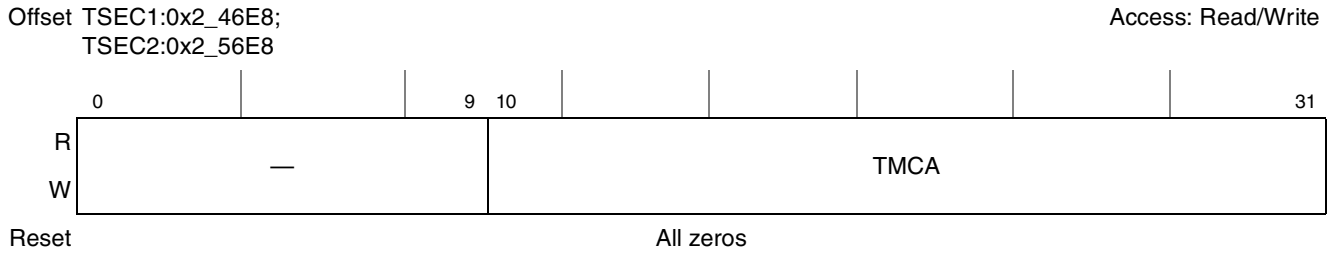


Figure 15-75. Transmit Multicast Packet Counter Register Definition

Table 15-73 describes the fields of the TMCA register.

Table 15-73. 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).

15.5.3.7.28 Transmit Broadcast Packet Counter Register (TBCA)

Figure 15-77 shows the TBCA register.

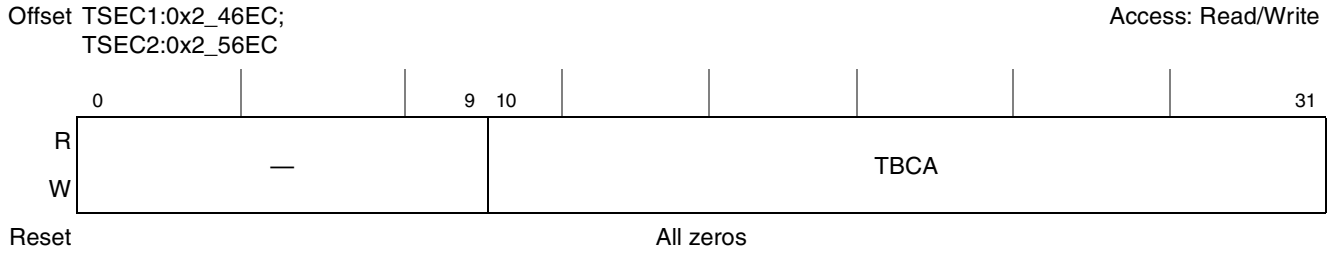


Figure 15-76. Transmit Broadcast Packet Counter Register Definition

Table 15-74 describes the fields of the TBCA register.

Table 15-74. 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).

15.5.3.7.29 Transmit Pause Control Frame Counter Register (TXPF)

Figure 15-78 shows the TXPF register.

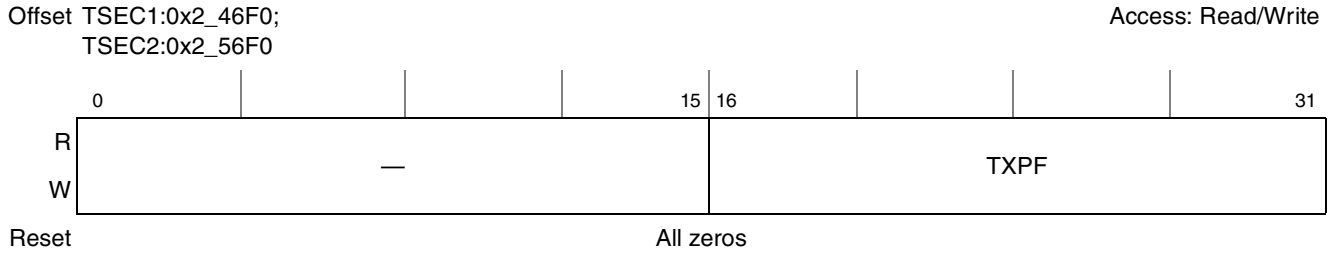


Figure 15-77. Transmit Pause Control Frame Counter Register Definition

Table 15-75 describes the fields of the TXPF register.

Table 15-75. 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.

15.5.3.7.30 Transmit Deferral Packet Counter Register (TDFR)

Figure 15-79 shows the TDFR register.

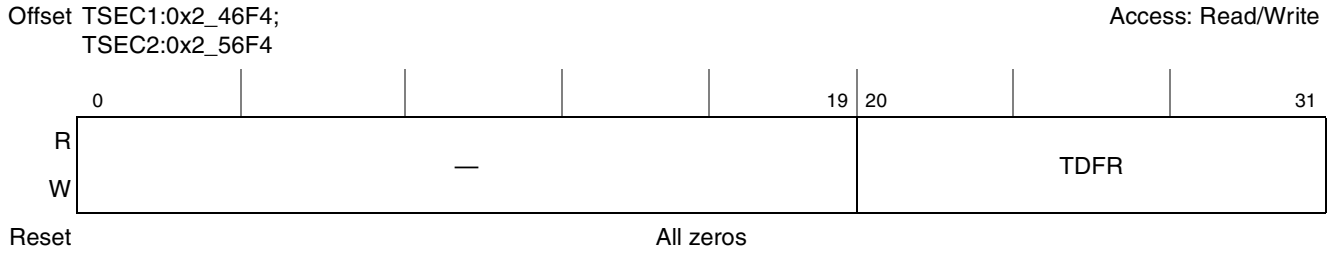


Figure 15-78. Transmit Deferral Packet Counter Register Definition

Table 15-76 describes the fields of the TDFR register.

Table 15-76. 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.

15.5.3.7.31 Transmit Excessive Deferral Packet Counter Register (TEDF)

Figure 15-80 shows the TEDF register.

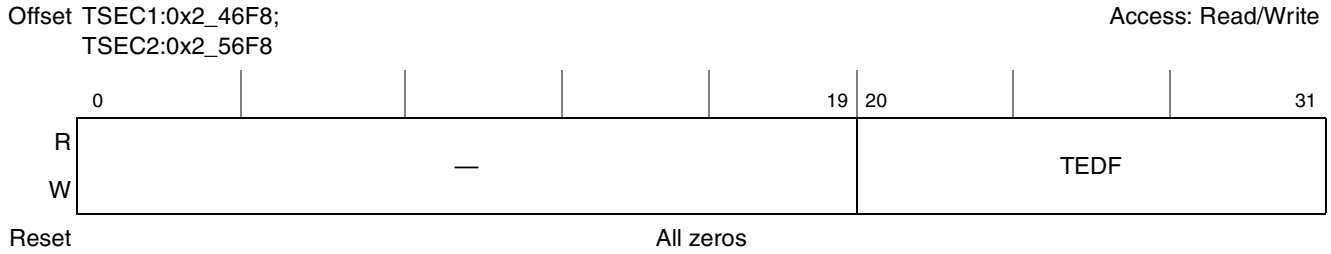


Figure 15-79. Transmit Excessive Deferral Packet Counter Register Definition

Table 15-77 describes the fields of the TEDF register.

Table 15-77. 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).

15.5.3.7.32 Transmit Single Collision Packet Counter Register (TSCL)

Figure 15-81 shows the TSCL register.

Offset TSEC1:0x2_46FC; TSEC2:0x2_56FC — Access: Read/Write

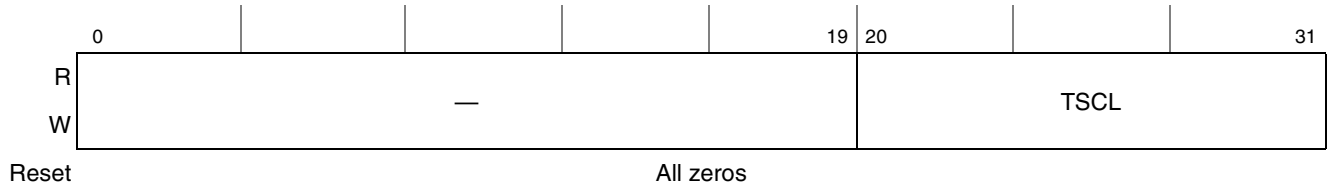


Figure 15-80. Transmit Single Collision Packet Counter Register Definition

Table 15-78 describes the fields of the TSCL register.

Table 15-78. 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.

15.5.3.7.33 Transmit Multiple Collision Packet Counter Register (TMCL)

Figure 15-82 shows the TMCL register.

Offset TSEC1:0x2_4700; TSEC2:0x2_5700 — Access: Read/Write

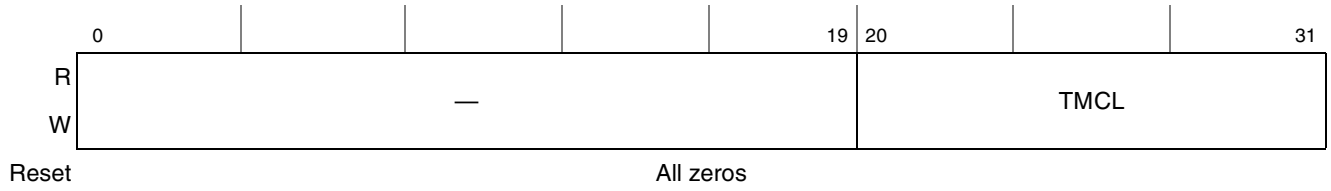


Figure 15-81. Transmit Multiple Collision Packet Counter Register Definition

Table 15-79 describes the fields of the TMCL register.

Table 15-79. 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.

15.5.3.7.34 Transmit Late Collision Packet Counter Register (TLCL)

Figure 15-83 shows the TLCL register.

Offset TSEC1:0x2_4704; TSEC2:0x2_5704 Access: Read/Write

	0	19	20	31
R	—		TLCL	
W				
Reset	All zeros			

Figure 15-82. Transmit Late Collision Packet Counter Register Definition

Table 15-80 describes the fields of the TLCL register.

Table 15-80. 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.

15.5.3.7.35 Transmit Excessive Collision Packet Counter Register (TXCL)

Figure 15-84 shows the TXCL register.

Offset TSEC1:0x2_4708; TSEC2:0x2_5708 Access: Read/Write

	0	19	20	31
R	—		TXCL	
W				
Reset	All zeros			

Figure 15-83. Transmit Excessive Collision Packet Counter Register Definition

Table 15-81 describes the fields of the TXCL register.

Table 15-81. 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.

15.5.3.7.36 Transmit Total Collision Counter Register (TNCL)

Figure 15-85 shows the TNCL register.

Offset TSEC1:0x2_470C; TSEC2:0x2_570C — Access: Read/Write

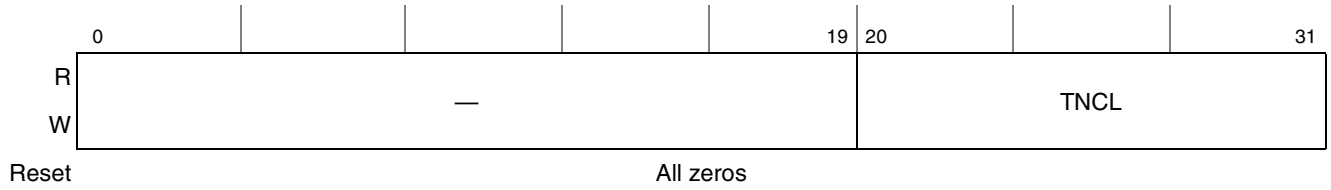


Figure 15-84. Transmit Total Collision Counter Register Definition

Table 15-82 describes the fields of the TNCL register.

Table 15-82. TNCL Field Descriptions

Bits	Name	Description
0–19	—	Reserved
20–31	TNCL	Transmit total collision counter. Increments by the number of collisions experienced during the transmission of a frame as defined as the simultaneous presence of signals on the DO and RD circuits (that is, transmitting and receiving at the same time). **Note:** This count does not include collisions that result in an excessive collision condition.

15.5.3.7.37 Transmit Drop Frame Counter Register (TDRP)

Figure 15-86 shows the TDRP register.

Offset TSEC1:0x2_4714; TSEC2:0x2_5714 — Access: Read/Write

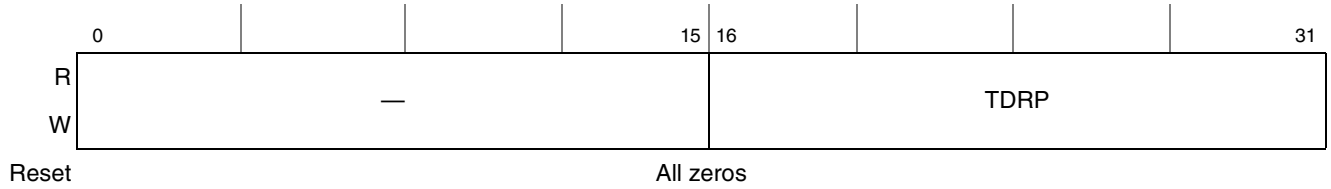


Figure 15-85. Transmit Drop Frame Counter Register Definition

Table 15-83 describes the fields of the TDRP register.

Table 15-83. 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.

15.5.3.7.38 Transmit Jabber Frame Counter Register (TJBR)

Figure 15-87 shows the TJBR register.

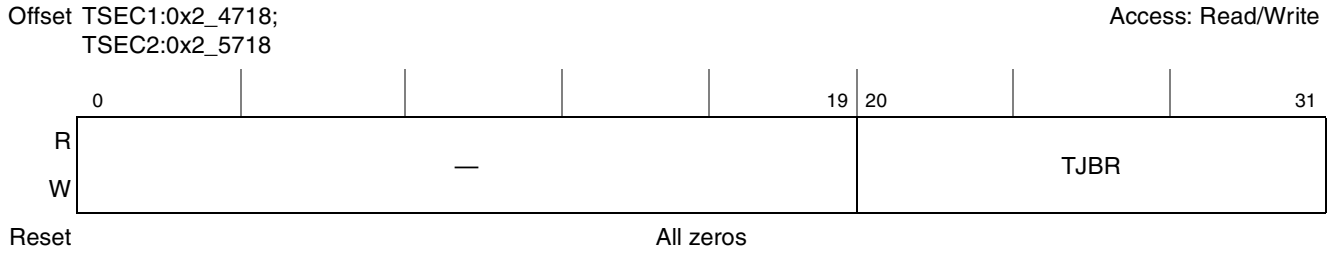


Figure 15-86. Transmit Jabber Frame Counter Register Definition

Table 15-84 describes the fields of the TJBR register.

Table 15-84. 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.

15.5.3.7.39 Transmit FCS Error Counter Register (TFCS)

Figure 15-88 shows the TFCS register.

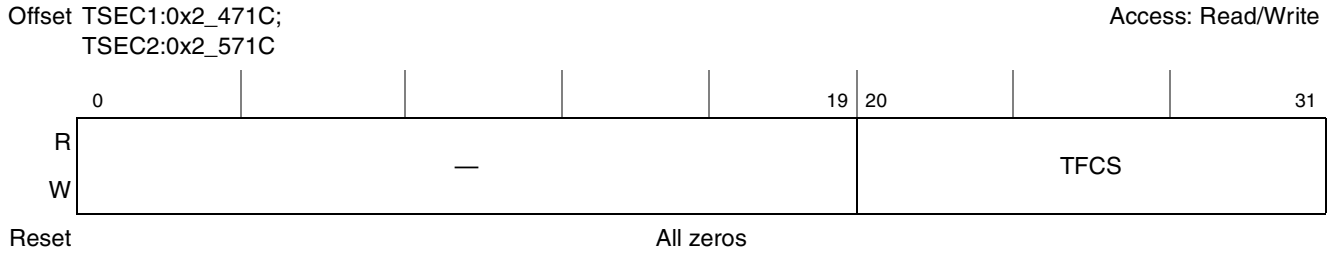


Figure 15-87. Transmit FCS Error Counter Register Definition

Table 15-85 describes the fields of the TFCS register.

Table 15-85. 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.

15.5.3.7.40 Transmit Control Frame Counter Register (TXCF)

Figure 15-89 shows the TXCF register.

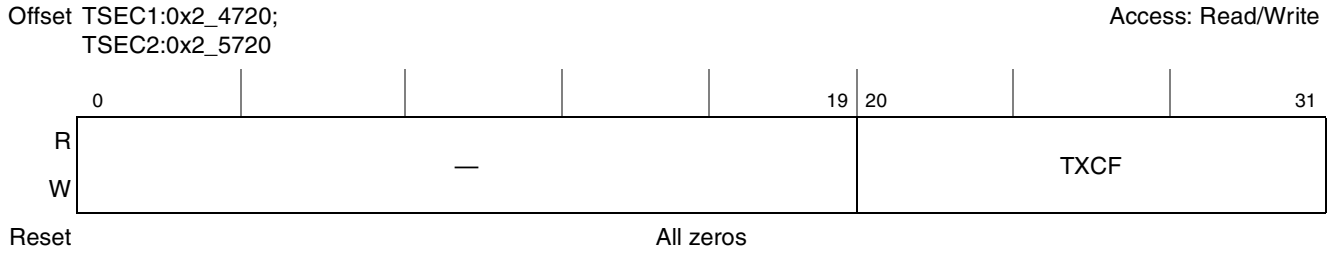


Figure 15-88. Transmit Control Frame Counter Register Definition

Table 15-86 describes the fields of the TXCF register.

Table 15-86. TXCF Field Descriptions

Bits	Name	Description
0–19	—	Reserved
20–31	TXCF	Transmit control frame counter. Increments for every valid size frame with a type field signifying a control frame.

15.5.3.7.41 Transmit Oversize Frame Counter Register (TOVR)

Figure 15-90 shows the TOVR register.

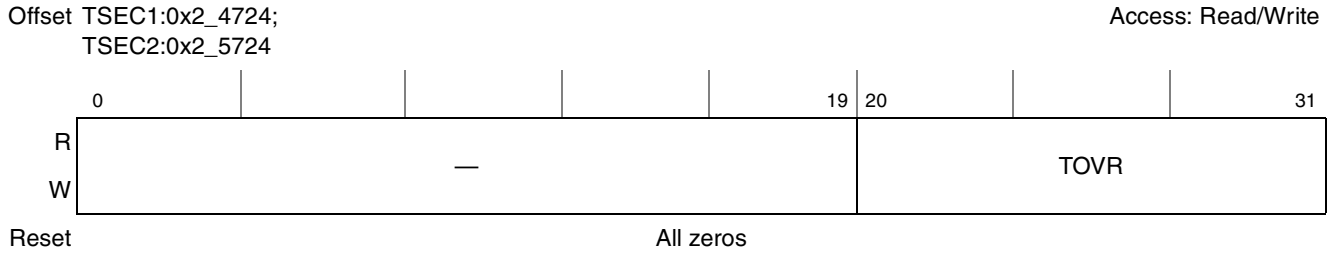


Figure 15-89. Transmit Oversized Frame Counter Register Definition

Table 15-87 describes the fields of the TOVR register.

Table 15-87. TOVR Field Descriptions

Bits	Name	Description
0–19	—	Reserved
20–31	TOVR	Transmit oversize frame counter. Increments for each oversized transmitted frame with a correct FCS value.

15.5.3.7.42 Transmit Undersize Frame Counter Register (TUND)

Figure 15-91 shows the TUND register.

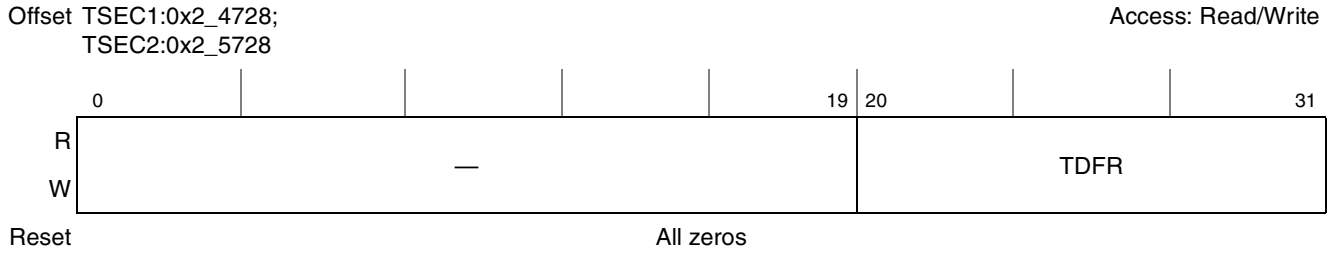


Figure 15-90. Transmit Undersize Frame Counter Register Definition

Table 15-88 describes the fields of the TUND register.

Table 15-88. TUND Field Descriptions

Bits	Name	Description
0–19	—	Reserved
20–31	TDFR	Transmit undersize frame counter. Increments for every frame less then 64 bytes, with a correct FCS value.

15.5.3.7.43 Transmit Fragment Counter Register (TFRG)

Figure 15-92 shows the TFRG register.

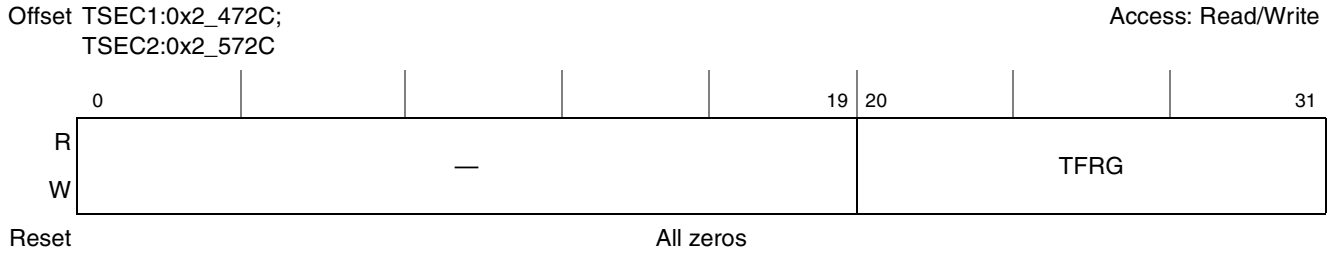


Figure 15-91. Transmit Fragment Counter Register Definition

Table 15-89 describes the fields of the TFRG register.

Table 15-89. TFRG Field Descriptions

Bits	Name	Description
0–19	—	Reserved
20–31	TFRG	Transmit fragment counter. Increments for every frame less then 64 bytes, with an incorrect FCS value.

15.5.3.7.44 Carry Register 1 (CAR1)

Carry register bits are cleared on carry register writes when the respective bits are set. Figure 15-93 shows the CAR1 register.

Offset TSEC1:0x2_4730; TSEC2:0x2_5730 — Access: Read/Write

	0	1	2	3	4	5	6	7
R W	C164	C1127	C1255	C1511	C11K	C1MAX	C1MGY	—
Reset	All zeros							

	14	15
R W	—	C1RBY
Reset	All zeros	

	21	22	23
R W	C1RXP	C1RXU	C1RAL
Reset	All zeros		

	24	25	26	27	28	29	30	31
R W	C1RFL	C1RCD	C1RCS	C1RUN	C1ROV	C1RFR	C1RJB	C1RDR
Reset	All zeros							

Figure 15-92. Carry Register 1 (CAR1) Register Definition

Table 15-90 describes the fields of the CAR1 register.

Table 15-90. 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
7–14	—	Reserved
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

Table 15-90. CAR1 Field Descriptions (continued)

Bits	Name	Description
20	C1RXC	Carry register 1 RXCF counter carry bit
21	C1RXP	Carry register 1 RXPF counter carry bit
22	C1RXU	Carry register 1 RXUO counter carry bit
23	C1RAL	Carry register 1 RALN counter carry bit
24	C1RFL	Carry register 1 RFLR counter carry bit
25	C1RCD	Carry register 1 RCDE counter carry bit
26	C1RCS	Carry register 1 RCSE counter carry bit
27	C1RUN	Carry register 1 RUND counter carry bit
28	C1ROV	Carry register 1 ROVR counter carry bit
29	C1RFR	Carry register 1 RFRG counter carry bit
30	C1RJB	Carry register 1 RJBR counter carry bit
31	C1RDR	Carry register 1 RDRP counter carry bit

15.5.3.7.45 Carry Register 2 (CAR2)

Figure 15-94 shows the CAR2 register.

Offset TSEC1:0x2_4734; TSEC2:0x2_5734 — Access: Read/Write

	0
R	—
W	
Reset	All zeros

	11	12	13	14	15
R	—	C2TJB	C2TFC	C2TCF	C2TOV
W					
Reset	All zeros				

	16	17	18	19	20	21	22	23
R	C2TUN	C2TFG	C2TBY	C2TPK	C2TMC	C2TBC	C2TPF	C2TDF
W								
Reset	All zeros							

	24	25	26	27	28	29	30	31
R	C2TED	C2TSC	C2TMA	C2TLC	C2TXC	C2TNC	—	C2TDP
W								
Reset	All zeros							

Figure 15-93. Carry Register 2 (CAR2) Register Definition

Carry register bits are cleared on carry register write when the respective bits are set. Table 15-91 describes the fields of the CAR2 register.

Table 15-91. CAR2 Field Descriptions

Bits	Name	Description
0–11	—	Reserved
12	C2TJB	Carry register 2 TJBR counter carry bit
13	C2TFC	Carry register 2 TFCS counter carry bit
14	C2TCF	Carry register 2 TXCF counter carry bit
15	C2TOV	Carry register 2 TOVR counter carry bit
16	C2TUN	Carry register 2 TUND counter carry bit
17	C2TFG	Carry register 2 TFRG counter carry bit
18	C2TBY	Carry register 2 TBYT counter carry bit
19	C2TPK	Carry register 2 TPKT counter carry bit
20	C2TMC	Carry register 2 TMCA counter carry bit
21	C2TBC	Carry register 2 TBCA counter carry bit
22	C2TPF	Carry register 2 TXPF counter carry bit
23	C2TDF	Carry register 2 TDFR counter carry bit
24	C2TED	Carry register 2 TEDF counter carry bit
25	C2TSC	Carry register 2 TSCL counter carry bit
26	C2TMA	Carry register 2 TMCL counter carry bit
27	C2TLC	Carry register 2 TLCL counter carry bit
28	C2TXC	Carry register 2 TXCL counter carry bit
29	C2TNC	Carry register 2 TNCL counter carry bit
30	—	Reserved
31	C2TDP	Carry register 2 TDRP counter carry bit

15.5.3.7.46 Carry Mask Register 1 (CAM1)

As long as one of the below mask bits is cleared, the corresponding interrupt bit is allowed to cause interrupt indications on output CARRY. These bits all default to a set state. Figure 15-95 shows the CAM1 register.

Offset TSEC1:0x2_4738; TSEC2:0x2_5738 Access: Read/Write

	0	1	2	3	4	5	6	7
R W	M164	M1127	M1255	M1511	M11K	M1MAX	M1MGV	—
Reset	1	1	1	1	1	1	1	0

	8	9	10	11	12	13	14	15
R W	—							M1RBY
Reset	0	0	0	0	0	0	0	1

	16	17	18	19	20	21	22	23
R W	M1RPK	M1RFC	M1RMC	M1RBC	M1RXC	M1RXP	M1RXU	M1RAL
Reset	1	1	1	1	1	1	1	1

	24	25	26	27	28	29	30	31
R W	M1RFL	M1RCD	M1RCS	M1RUN	M1ROV	M1RFR	M1RJB	M1RDR
Reset	1	1	1	1	1	1	1	1

Figure 15-94. Carry Mask Register 1 (CAM1) Register Definition

Table 15-92 describes the fields of the CAM1 register.

Table 15-92. CAM1 Field Descriptions

Bits	Name	Description
0	M164	Mask register 1 TR64 counter carry bit mask
1	M1127	Mask register 1 TR127 counter carry bit mask
2	M1255	Mask register 1 TR255 counter carry bit mask
3	M1511	Mask register 1 TR511 counter carry bit mask
4	M11K	Mask register 1 TR1K counter carry bit mask
5	M1MAX	Mask register 1 TRMAX counter carry bit mask
6	M1MGV	Mask register 1 TRMGV counter carry bit mask
7–14	—	Reserved
15	M1RBY	Mask register 1 RBYT counter carry bit mask
16	M1RPK	Mask register 1 RPKT counter carry bit mask
17	M1RFC	Mask register 1 RFCS counter carry bit mask
18	M1RMC	Mask register 1 RMCA counter carry bit mask

Table 15-92. CAM1 Field Descriptions (continued)

Bits	Name	Description
19	M1RBC	Mask register 1 RBCA counter carry bit mask
20	M1RXC	Mask register 1 RXCF counter carry bit mask
21	M1RXP	Mask register 1 RXPF counter carry bit mask
22	M1RXU	Mask register 1 RXUO counter carry bit mask
23	M1RAL	Mask register 1 RALN counter carry bit mask
24	M1RFL	Mask register 1 RFLR counter carry bit mask
25	M1RCD	Mask register 1 RCDE counter carry bit mask
26	M1RCS	Mask register 1 RCSE counter carry bit mask
27	M1RUN	Mask register 1 RUND counter carry bit mask
28	M1ROV	Mask register 1 ROVR counter carry bit mask
29	M1RFR	Mask register 1 RFRG counter carry bit mask
30	M1RJB	Mask register 1 RJBR counter carry bit mask
31	M1RDR	Mask register 1 RDRP counter carry bit mask

15.5.3.7.47 Carry Mask Register 2 (CAM2)

As long as one of the below mask bits is cleared, the corresponding interrupt bit is allowed to cause interrupt indications on output CARRY. These bits default to a set state. Figure 15-96 shows the CAM2 register.

Offset TSEC1:0x2_473C; TSEC2:0x2_573C — Access: Read/Write

	0							
R W	—							
Reset	All zeros							

	11	12	13	14	15
R W	—	M2TJB	M2TFC	M2TCF	M2TOV
Reset	0 0 0 0	1	1	1	1

	16	17	18	19	20	21	22	23
R W	M2TUN	M2TFG	M2TBY	M2TPK	M2TMC	M2TBC	M2TPF	M2TDF
Reset	1	1	1	1	1	1	1	1

	24	25	26	27	28	29	30	31
R W	M2TED	M2TSC	M2TMA	M2TLC	M2TXC	M2TNC	—	M2TDP
Reset	1	1	1	1	1	1	0	1

Figure 15-95. Carry Mask Register 2 (CAM2) Register Definition

Table 15-93 describes the fields of the CAM2 register.

Table 15-93. 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
30	—	Reserved
31	M2TDP	Mask register 2 TDRP counter carry bit mask

15.5.3.8 Hash Function Registers

This section provides detailed descriptions of the registers used for hash functions. All of the registers are 32 bits wide. If the DA field of a receive frame is processed through a 32-bit CRC generator, the 8 bits of the CRC remainder is mapped to a hash table entry. The user can enable a hash entry by setting the appropriate 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. The user must further filter the address. See Section 15.6.2.6.2, “Hash Table Algorithm”.

15.5.3.8.1 Individual Address Registers 0–7 (IADDR*n*)

The IADDR*n* registers, shown in Figure 15-97, are written by the user. These registers represent 256 entries of the individual (unicast) address hash table used in the address recognition process. When the DA field of a receive frame is processed through a 32-bit CRC generator, the 8 high-order bits (0–7) of the

CRC remainder are mapped to one of the 256 entries. 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.

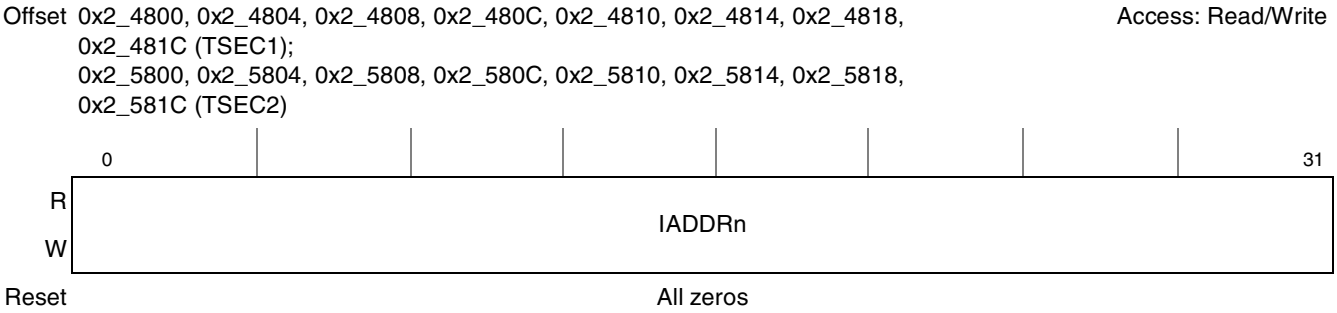


Figure 15-96. IADDR*n* Register Definition

Table 15-94 describes the field of the IADDR*n* registers.

Table 15-94. IADDR*n* Field Description

Bits	Name	Description
0–31	IADDR*n*	Represents the 32-bit value associated with the corresponding register. IADDR0 contains the high-order 32 bits of the 256-entry hash table and IADDR7 represents the low-order 32 bits. For instance, the msb of IADDR0 correlates to entry 0 and the lsb of IADDR7 correlates to entry 255.

15.5.3.8.2 Group Address Registers 0–7 (GADDR*n*)

The GADDR*n* registers are written by the user. Together these registers represent 256 entries of the group (multicast) address hash table used in the address recognition process. While the DA field of a receive frame is processed through a 32-bit CRC generator, the 8 bits of the CRC remainder is mapped to one of the 256 entries. 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 15-98 shows the GADDR*n* registers.

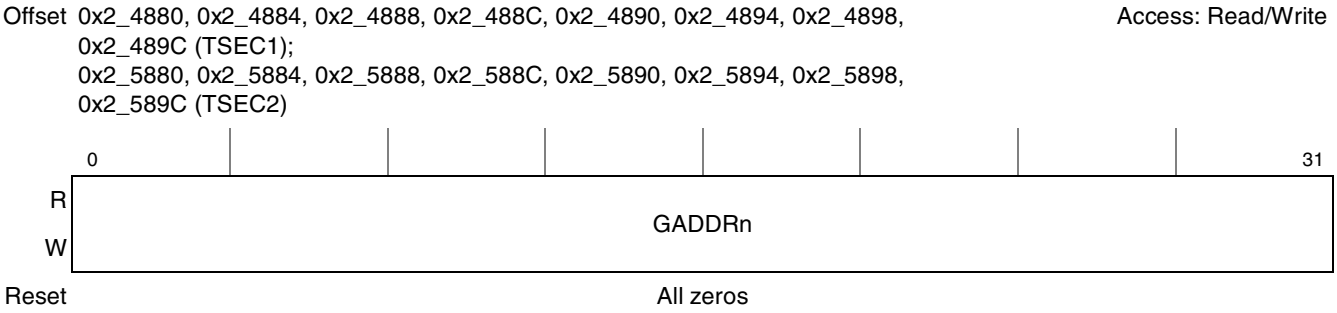


Figure 15-97. GADDR*n* Register Definition

Table 15-95 describes GADDR*n*.

Table 15-95. GADDR*n* Field Description

Bits	Name	Description
0–31	GADDR*n*	Represents the 32-bit value associated with the corresponding register. GADDR0 contains the high-order 32 bits of the 256 entry group hash table and GADDR7 represents the low-order 32 bits. For instance, the msb of GADDR0 correlates to entry 0 and the lsb of GADDR7 correlates to entry 255.

15.5.3.9 Attribute Registers

This section describes the two TSEC frame attribute registers.

15.5.3.9.1 Attribute Register (ATTR)

The attribute register, shown in Figure 15-99, defines 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.

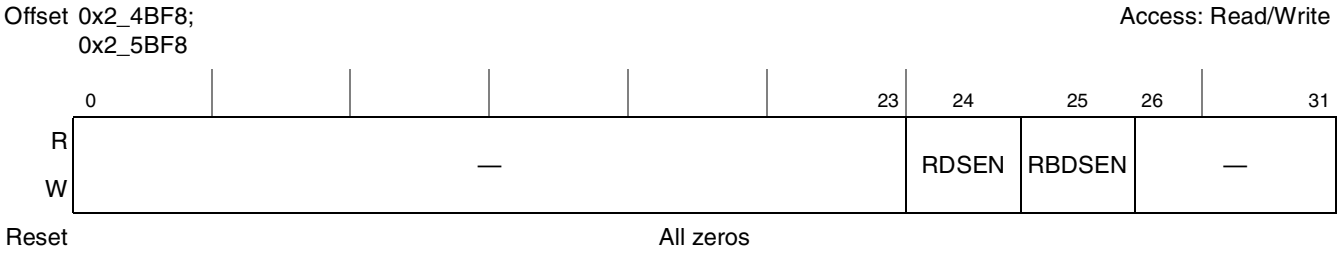


Figure 15-98. ATTR Register Definition

Table 15-96 describes the fields of the ATTR register.

Table 15-96. ATTR Field Descriptions

Bits	Name	Description
0–23	—	Reserved
24	RDSEN	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	RBDSEN	RxBD snoop enable. 0 Disables snooping of all receive BD memory accesses. 1 Enables snooping of all receive BD memory accesses.
26–31	—	Reserved

15.5.4 Ten-Bit Interface (TBI)

This section describes the ten-bit interface (TBI) and the TBI MII set of registers.

15.5.4.1 TBI MII Set Register Descriptions

This section describes the TBI MII registers. All TBI registers are 16 bits wide and are accessed at the offset of the TBI physical address. The TBI physical address of the TSEC is stored in TBIPA. Writing to the TBI registers is similar to writing to an external PHY, by using the MII management interface. By using TBIPA in place of the PHY address, in the MIIMADD[PHY Address] field, and setting MIIMADD[Register Address] to the address offset that corresponds to the desired register (see Table 15-97), the user can read (set MIIMCOM[Read Cycle]) or write (writing to MIIMCON[PHY control]) to the TBI block. Refer to the TBI physical address register in Section 15.5.3.1, "TSEC General Control and Status Registers," and the TBI MII register set in Table 15-97.

Note that jitter diagnostics and TBI control are not IEEE 802.3 required registers and are used only for test and control of the TSEC TBI block. The TBI's TBI control register (TBI) is for configuring the TSEC ten-bit interface block. However, because this TBI block has an MII management interface (just like any other PHY), it has an IEEE 802.3 register called the control register (CR).

Table 15-97. TBI MII Register Set

Offset	Name	Access	Size	Section/Page
TEN-BIT INTERFACE (TBI) REGISTERS				
0x00	Control (CR)	R/W[1]	16 bits	15.5.4.2/15-91
0x01	Status (SR)	R, LH, LL	16 bits	15.5.4.3/15-92
0x02–0x03	Reserved	R	2 x 16 bits	—
0x04	AN advertisement (ANA)	RW, R	16 bits	15.5.4.3/15-92
0x05	AN link partner base page ability (ANLPBPA)	R	16 bits	15.5.4.5/15-95
0x06	AN expansion (ANEX)	R, LH	16 bits	15.5.4.6/15-96
0x07	AN next page transmit (ANNPT)	R/W, R	16 bits	15.5.4.7/15-97
0x08	AN link partner ability next page (ANLPANP)	R	16 bits	15.5.4.8/15-98
0x0F	Extended status (EXST)	R	16 bits	15.5.4.9/15-98
0x10	Jitter diagnostics (JD)	R/W	16 bits	15.5.4.10/15-99
0x11	TBI control (TBICON)	R/W	16 bits	15.5.4.11/15-100

[1] R = means read only, WO = write only, R/W = read and write, LH = latches high, LL = latches low, SC = self-clearing,

15.5.4.2 Control Register (CR)

Figure 15-100 shows the CR register.

Offset TBIPA offset + 0x00 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	15
R W	PHY Reset	—	Speed_0	AN Enable	—		Reset AN	Full Duplex	—	Speed_1	—	
Reset	0	0	0	1	0	0	0	1	0	0	0 0	0 0 0 0

Figure 15-99. Control Register Definition

Table 15-98 describes the fields of the CR register.

Table 15-98. CR Field Descriptions

Bits	Name	Description
0	PHY Reset	PHY reset. This bit is cleared by default. This bit is self-clearing. 0 Normal operation. 1 The control and status registers are returned to their default value. This in turn may change the internal state of the TBI and its link partner.
1	—	Reserved
2	Speed_0	Speed selection. This bit defaults to a cleared state and must stay cleared, which corresponds to 1-Gbps speed.
3	AN Enable	Auto-negotiation enable. This bit is set by default. 0 The values programmed in bits 2, 7, and 9 determine the operating condition of the link. 1 Auto-negotiation process enabled.
4–5	—	Reserved
6	Reset AN	Reset auto-negotiation. This bit is cleared by default and is self-clearing. 0 Normal operation. 1 The auto-negotiation process restarts. This action is only available if auto-negotiation is enabled.
7	Full Duplex	Duplex mode. This bit is set by default. 0 Half-duplex operation. 1 Full-duplex operation.
8	—	Reserved, must be cleared.
9	Speed_1	Speed selection. **Maximum Operating Speed** / **Bit 2** / **Bit 9** Reserved / 0 / 0 Reserved / 1 / 0 1 Gbps / 0 / 1 Reserved / 1 / 1
10–15	—	Reserved

15.5.4.3 Status Register (SR)

Figure 15-101 shows the SR register.

Offset TBIPA offset + 0x01 Access: Read Only

	0						6	7	8	9	10	11	12	13	14	15
R	—							Extend Status	—	No Pre	AN Done	Remote Fault	AN Ability	Link Status	—	Extend Ability
W																
Reset	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	1

Figure 15-100. Status Register Definition

Table 15-99 describes the fields of the SR register.

Table 15-99. SR Descriptions

Bits	Name	Description
0–6	—	Reserved, should be cleared.
7	Extend Status	Indicates that PHY status information is also contained in the extended status register. Returns 1 on read. This bit is read-only.
8	—	Reserved, must be cleared.
9	No Pre	MF preamble suppression enable. This bit indicates whether the PHY is capable of handling MII management frames without the 32-bit preamble field. Returns 1, indicating support for suppressed preamble MII management frames. This bit is read-only.
10	AN Done	Auto-negotiation complete. This bit is read-only and is cleared by default. 0 Either the auto-negotiation process is underway or the auto-negotiation function is disabled. 1 The auto-negotiation process has completed.
11	Remote Fault	Remote fault. This bit is read-only and is cleared by default. Each read of SR clears this bit. 0 Normal operation. 1 A remote fault condition was detected. Latches high for software to detect the condition.
12	AN Ability	Auto-negotiation ability. While read as set, this bit indicates that the PHY has the ability to perform auto-negotiation. While read as cleared, this bit indicates the PHY lacks the ability to perform auto-negotiation. Returns 1 on read. This bit is read-only.
13	Link Status	Link status. This bit is read-only and is cleared by default. 0 A valid link is not established. Latches low allowing for software polling to detect a failure condition. 1 A valid link is established.
14	—	Reserved, must be cleared.
15	Extend Ability	Extended capability. This bit indicates that the PHY contains the extended set of registers (those beyond control and status). Returns 1 on read. This bit is read-only.

15.5.4.4 AN Advertisement Register (ANA)

Figure 15-102 shows the ANA register.

Offset TBIPA offset + 0x04 Access: Read/Write

	0	1	2 3	4 6	7	8	9	10	11 15
R	Next Page	Ack	Remote Fault	—	Pause		Half Duplex	Full Duplex	—
W									
Reset	All zeros								

Figure 15-101. AN Advertisement Register Definition

Table 15-100 describes the fields of the ANA register.

Table 15-100. ANA Field Descriptions

Bits	Name	Description
0	Next Page	Next page configuration. The local device sets this bit to either request next page transmission or advertise next page exchange capability. 0 The local device wishes not to engage in next page exchange. 1 The local device has no next pages but wishes to allow reception of next pages. If the local device has no next pages and the link partner wishes to send next pages, the local device shall send null message codes and have the message page set to 0b000_0000_0001, as defined in annex 28C of the *IEEE 802.3 specification*.
1	Ack	Acknowledge. Write 0, ignore on read.This bit is read-only. (Reserved)
2–3	Remote Fault	The local device's remote fault condition is encoded in bits 2 and 3 of the base page. Values are shown in the following table. The default value is 00. Indicate a fault by setting a non-zero remote fault encoding and re-negotiating.
4–6	—	Reserved, must be cleared.

RF1 bit[3]	RF2 bit[2]	Description
0	0	No error, link OK
0	1	Offline
1	0	Link_Failure
1	1	Auto-Negotiation_Error

Table 15-100. ANA Field Descriptions (continued)

Bits	Name	Description
7–8	Pause	The local device's PAUSE capability is encoded in bits 7 and 8, and the decodes are shown in the following table. For priority resolution information consult Table 15-101.
9	Half Duplex	Half-duplex capability 0 Designates local device as not capable of half-duplex operation. 1 Designates local device as capable of half-duplex operation.
10	Full Duplex	Full-duplex capability 0 Designates the local device as not capable of full-duplex operation. 1 Designates the local device as capable of full-duplex operation.
11–15	—	Reserved, must be cleared.

PAUSE bit[8]	ASM_DIR bit[7]	Capability
0	0	No PAUSE
0	1	Asymmetric PAUSE toward link partner
1	0	Symmetric PAUSE
1	1	Both symmetric PAUSE and Asymmetric PAUSE toward local device

Table 15-101 describes the resolution of pause priority.

Table 15-101. PAUSE Priority Resolution

Local Device		Link Parter		Local Resolution	Link Partner Resolution
PAUSE	ASM_DIR	PAUSE	ASM_DIR		
0	0	x	x	Disable PAUSE transmit Disable PAUSE receive	Disable PAUSE transmit Disable PAUSE receive
0	1	0	x	Disable PAUSE transmit Disable PAUSE receive	Disable PAUSE transmit Disable PAUSE receive
0	1	1	0	Disable PAUSE transmit Disable PAUSE receive	Disable PAUSE transmit Disable PAUSE receive
0	1	1	1	Enable PAUSE transmit Disable PAUSE receive	Disable PAUSE transmit Enable PAUSE receive
1	0	0	x	Disable PAUSE transmit Disable PAUSE receive	Disable PAUSE transmit Disable PAUSE receive
1	0	1	x	Enable PAUSE transmit Enable PAUSE receive	Enable PAUSE transmit Enable PAUSE receive
1	1	0	0	Disable PAUSE transmit Disable PAUSE receive	Disable PAUSE transmit Disable PAUSE receive

Table 15-101. PAUSE Priority Resolution (continued)

Local Device		Link Parter		Local Resolution	Link Partner Resolution
PAUSE	ASM_DIR	PAUSE	ASM_DIR		
1	1	0	1	Disable PAUSE transmit Enable PAUSE receive	Enable PAUSE transmit Disable PAUSE receive
1	1	1	x	Enable PAUSE transmit Enable PAUSE receive	Enable PAUSE transmit Enable PAUSE receive

15.5.4.5 AN Link Partner Base Page Ability Register (ANLPBPA)

Figure 15-103 shows the ANLPBPA register.

Offset TBIPA offset + 0x05 Access: Read Only

	0	1	2	3	4 — 6	7	8	9	10	11 — 15
R	Next Page	Ack	Remote Fault		—	Pause		Half Duplex	Full Duplex	—
W										
Reset	All zeros									

Figure 15-102. AN Link Partner Base Page Ability Register Definition

Table 15-102 describes the fields of the ANLPBPA register.

Table 15-102. ANLPBPA Field Descriptions

Bits	Name	Description
0	Next Page	Next page. This bit is read-only. The link partner sets or clears this bit. 0 Link partner has no subsequent next pages or is not capable of receiving next pages. 1 Link partner either requesting next page transmission or indicating the capability to receive next pages.
1	Ack	Acknowledge. Ignore on read. This bit is read-only
2–3	Remote Fault	The link partner's remote fault condition is encoded in bits 2 and 3 of the base page. Values are shown in the remote fault encoding field table below. This bit is read-only.
4–6	—	Reserved, must be cleared.

RF1 bit[3]	RF2 bit[2]	Description
0	0	No error, link OK
0	1	Offline
1	0	Link_Failure
1	1	Auto-Negotiation_Error

Table 15-102. ANLPBPA Field Descriptions (continued)

Bits	Name	Description
7–8	Pause	Encoding of the link partner's PAUSE capability is shown in the PAUSE encoding table below. For priority resolution information consult the IEEE 802.3 specification. This bit is read-only **PAUSE bit[8] / ASM_DIR bit[7] / Capability** 0 / 0 / No PAUSE 0 / 1 / Asymmetric PAUSE toward link partner 1 / 0 / Symmetric PAUSE 1 / 1 / Both symmetric PAUSE and Asymmetric PAUSE toward local device
9	Half Duplex	Half-duplex capability. This bit is read-only. 0 Link partner is not capable of half-duplex mode. 1 Link partner is capable of half-duplex mode.
10	Full Duplex	Full-duplex capability. This bit is read-only. 0 Link partner is not capable of full-duplex mode. 1 Link partner is capable of full-duplex mode.
11–15	—	Reserved, must be cleared.

15.5.4.6 AN Expansion Register (ANEX)

Figure 15-104 shows the ANEX register.

Offset TBIPA offset + 0x06 Access: Read Only

	0												12	13	14	15
R	—													NP Able	Page Rx'd	—
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Figure 15-103. AN Expansion Register Definition

Table 15-103 describes the fields of the ANEX register.

Table 15-103. ANEX Field Descriptions

Bits	Name	Description
0–12	—	Reserved, must be cleared.
13	NP Able	Next page able. This bit is read-only and returns 1 on read. While read as set, indicates local device supports next page function.

Table 15-103. ANEX Field Descriptions (continued)

Bits	Name	Description
14	Page Rx'd	Page received. This bit is read-only. The bit clears on a read to the register. 0 Normal operation. 1 A new page was received and stored in the applicable AN link partner ability or AN next page register. This bit latches high in order for software to detect while polling.
15	—	Reserved, must be cleared.

15.5.4.7 AN Next Page Transmit Register (ANNPT)

Figure 15-105 shows the ANNPT register.

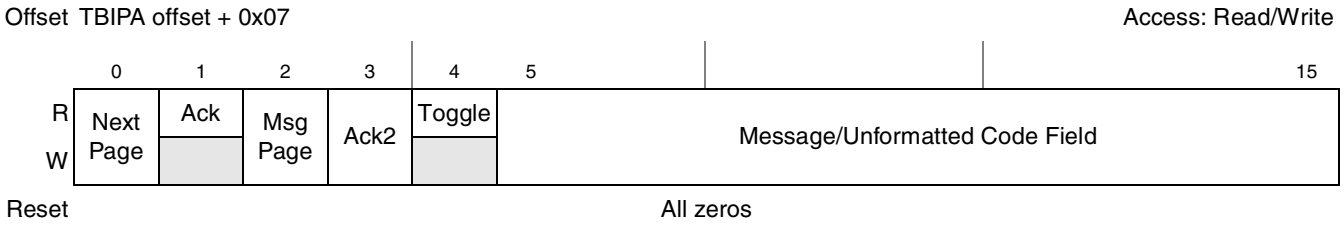


Figure 15-104. AN Next Page Transmit Register Definition

Table 15-104 describes the fields of the ANNPT register.

Table 15-104. ANNPT Field Descriptions

Bits	Name	Description
0	Next Page	Next page indication. [Reference MII bit 7.15 in IEEE 802.3, 2000 Edition Clause 28.2.4] 0 Last page 1 Additional next pages to follow
1	Ack	Acknowledge. Write 0, ignore on read. [Reference MII bit 7.14] This bit is read-only.
2	Msg Page	Message page. [Reference MII bit 7.13] 0 Unformatted page 1 Message page
3	Ack2	Acknowledge 2. Used by the next page function to indicate that the device has the ability to comply with the message. [Reference MII bit 7.12] 0 The local device cannot comply with message. 1 The local device complies with message.
4	Toggle	Toggle. Used to ensure synchronization with the link partner during next page exchange. This bit always takes the opposite value of the toggle bit of the previously-exchanged link code word. The initial value in the first next page transmitted is the inverse of bit 11 in the base link code word. [Reference MII bit 7.11] This bit is read-only. 0 Toggle bit of the previously-exchanged link code word was set. 1 Toggle bit of the previously-exchanged link code word was clear.
5–15	Message/ Unformatted Code Field	Message pages are formatted pages that carry a pre-defined message code, which is enumerated in IEEE 802.3u/Annex 28C. Unformatted code fields take on an arbitrary value. [Reference MII field 7.10:0]

15.5.4.8 AN Link Partner Ability Next Page Register (ANLPANP)

Figure 15-106 shows the ANLPANP register.

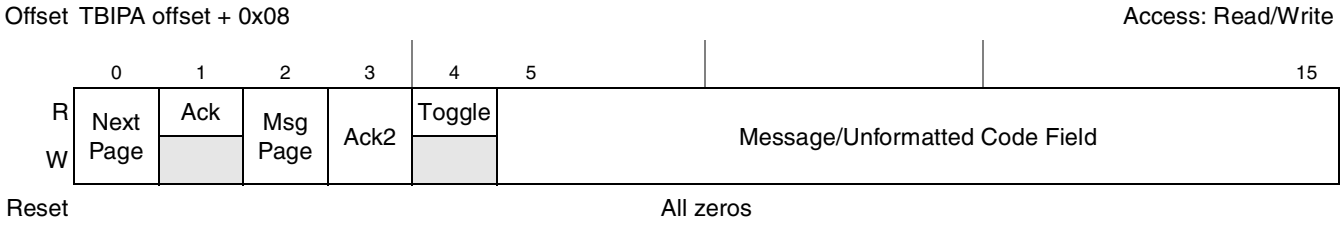


Figure 15-105. AN Link Partner Ability Next Page Register Definition

Table 15-105 describes the ANLPANP fields.

Table 15-105. ANLPANP Field Descriptions

Bits	Name	Description
0	Next Page	Next page. The link partner sets and clears this bit. 0 Last page from link partner. 1 Additional [next] pages to follow.
1	Ack	Acknowledge. Ignore on read. [Reference MII bit 8.14] This bit is read-only.
2	Msg Page	Message page 0 Unformatted page. 1 Message page.
3	Ack2	Acknowledge 2. Indicates the link partner's ability to comply with the message. 0 Link partner cannot comply with message. 1 Link partner complies with message.
4	Toggle	Toggle. Used to ensure synchronization with the link partner during next page exchange. This bit always takes the opposite value of the toggle bit of the previously-exchanged link code word. The initial value in the first next page transmitted is the inverse of bit 11 in the base link code word. This bit is read-only. 0 Toggle bit of the previously-exchanged link code word was set. 1 Toggle bit of the previously-exchanged link code word was clear.
5–15	Message/ Unformatted Code Field	Message pages are formatted pages that carry a pre-defined message code, which is enumerated in IEEE 802.3u/Annex 28C. Unformatted code fields take on an arbitrary value.

15.5.4.9 Extended Status Register (EXST)

Figure 15-107 shows the EXST register.

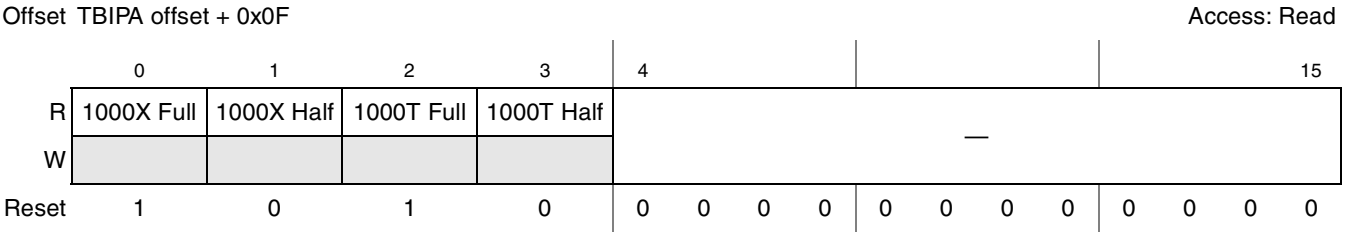


Figure 15-106. Extended Status Register Definition

Table 15-106 describes the fields of the EXST register.

Table 15-106. EXST Field Descriptions

Bits	Name	Description
0	1000X Full	1000X full-duplex capability. Returns 1 on read. This bit is read-only. 0 PHY cannot operate in 1000BASE-X full-duplex mode. 1 PHY can operate in 1000BASE-X full-duplex mode.
1	1000X Half	1000X half-duplex capability. Returns 0 on read. This bit is read-only. 0 PHY cannot operate in 1000BASE-X half-duplex mode. 1 PHY can operate in 1000BASE-X half-duplex mode.
2	1000T Full	1000T full-duplex capability. Returns 1 on read. This bit is read-only. 0 PHY cannot operate in 1000BASE-T full-duplex mode. 1 PHY can operate in 1000BASE-T full-duplex mode.
3	1000T Half	1000T half-duplex capability. Returns 0 on read. This bit is read-only. 0 PHY cannot operate in 1000BASE-T half-duplex mode. 1 PHY can operate in 1000BASE-T half-duplex mode.
4–15	—	Reserved

15.5.4.10 Jitter Diagnostics Register (JD)

Annex 36A in IEEE 802.3z describes several jitter test patterns. These can be configured to be sent by writing the jitter diagnostics register. See the register description for more information. It may be wise to auto-negotiate and advertise a remote fault, signaling offline, prior to beginning the test patterns. Figure 15-108 shows the JD register.

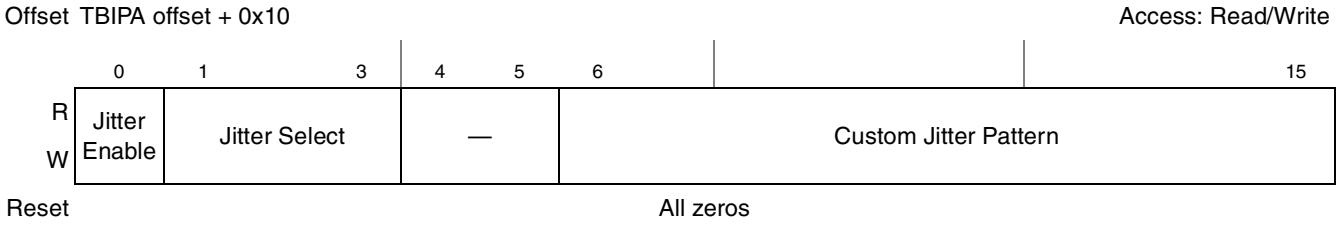


Figure 15-107. Jitter Diagnostics Register Definition

Table 15-107 describes the fields of the JD register.

Table 15-107. JD Field Description

Bits	Name	Description
0	Jitter Enable	Jitter enable. This bit is cleared by default. 0 Normal transmit operation. 1 Enable the TBI to transmit the jitter test patterns defined in annex 36A of the *IEEE 802.3 specification*.
1–3	Jitter Select	Selects the jitter pattern to be transmitted in diagnostics mode. Encoding of this field is shown in the following table. Default is 0.

Jitter Pattern Select	bit[1]	bit[2]	bit[3]
User defined uses custom jitter pattern, bits 6–15	0	0	0
High frequency (±D21.5) 1010_1010_1010_1010_1010_1010_1010_1010_1010_1010…	0	0	1
Mixed frequency (±K28.5) 1111_1010_1100_0001_0100_1111_1010_1100_0001_0100…	0	1	0
Low frequency 1111_1000_0011_1110_0000_1111_1000_0011_1110_0000…	0	1	1
Complex pattern (10'h17c,10'h0c9,10'h0e5,10'h2a3, 10'h17c,...)	1	0	0
Square Wave (-K28.7) 0011_1110_0000_1111_1000_0011_1110_0000_1111_1000…	1	0	1
Reserved	1	1	0
Reserved	1	1	1

Bits	Name	Description
4–5	—	Reserved
6–15	Custom Jitter Pattern	Used in conjunction with jitter (pattern) select and jitter (diagnostic) enable; set this field to the desired custom pattern which is continuously transmitted. Its default is 0.

15.5.4.11 TBI Control Register (TBICON)

Figure 15-109 shows the TBICON register.

Offset TBIPA offset + 0x11 Access: Read/Write

	0	1	2	3	4 6	7	8 9	10	11	12 15
R W	Soft Reset	—	Disable Rx Dis	Disable Tx Dis	—	AN Sense	—	Clock Select	MII Mode	—
Reset	All zeros									

Figure 15-108. TBI Control Register Definition

Table 15-108 describes the fields of the TBICON register.

Table 15-108. TBICON Field Description

Bits	Name	Description
0	Soft_Reset	Soft reset. This bit is cleared by default. 0 Normal operation. 1 Resets the functional modules in theTBI.
1	—	Reserved. (Ignore on read)
2	Disable Rx Dis	Disable receive disparity. This bit is cleared by default. 0 Normal operation. 1 Disables the running disparity calculation and checking in the receive direction.
3	Disable Tx Dis	Disable transmit disparity. This bit is cleared by default. 0 Normal operation. 1 Disables the running disparity calculation and checking in the transmit direction.
4–6	—	Reserved
7	AN Sense	Auto-negotiation sense enable. This bit is cleared by default. 0 IEEE 802.3z Clause 37 behavior is desired, which results in the link not completing. 1 Allow the auto-negotiation function to sense either a Gigabit MAC in auto-negotiation bypass mode or an older Gigabit MAC without auto-negotiation capability. If sensed, auto-negotiation complete becomes true; however, the page received is low, indicating no page was exchanged. Management can then act accordingly.
8–9	—	Reserved
10	Clock Select	Clock select. This bit is cleared by default. 0 Allow the TBI to accept dual split-phase 62.5-MHz receive clocks. 1 Configure the TBI to accept a 125-MHz receive clock from the SerDes/PHY. The 125-MHz clock must be physically connected to PMA receive clock 0.
11	MII Mode	This bit describes the configuration mode of the TBI. The user reads a 1 while the TBI is configured in GMII/MII mode (connected to a GMII/MII PHY) and a 0 while configured in TBI mode (connected to a 1000BASE-X SerDes). Its value is the inverse of ECNTRL[TBIM]. 0 TBI mode. 1 GMII mode.
12–13	—	Reserved
14–15	—	Reserved

15.6 Functional Description

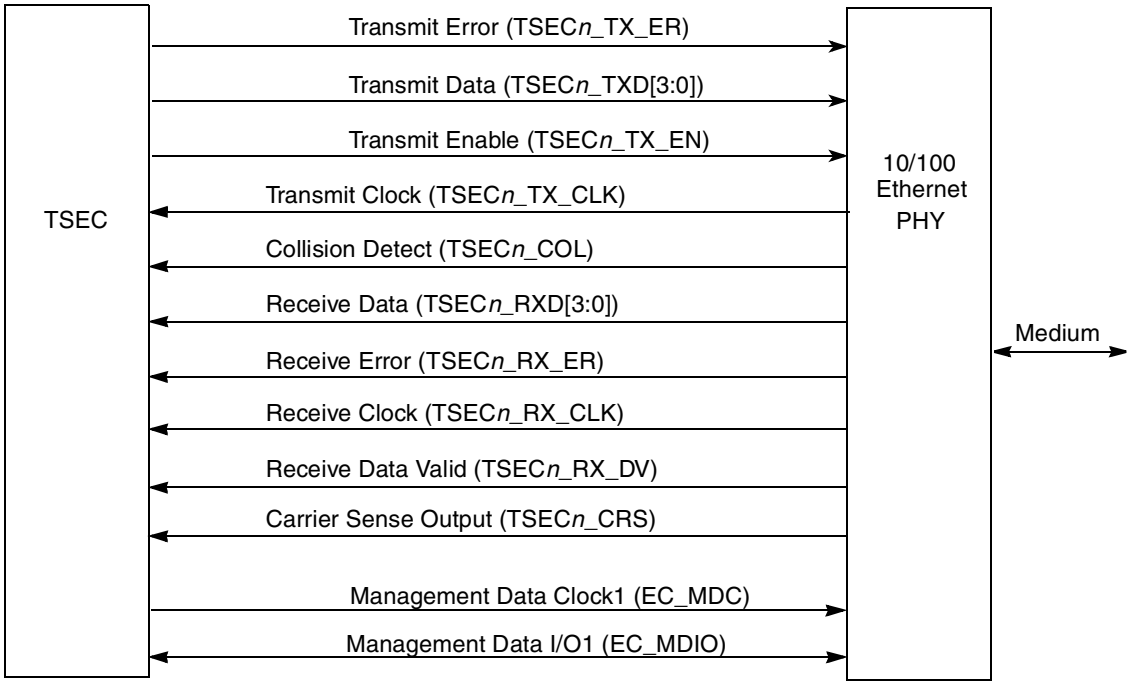
This section describes many of the functions of the TSEC controller.

15.6.1 Connecting to Physical Interfaces

This section describes how to connect the TSEC to various interfaces: MII, GMII, RGMII, RTBI and TBI. To avoid confusion, all of the buses follow the bus conventions used in the IEEE 802.3 specification, because each PHY follows the same convention. (For instance, in the bus TSEC*n*_TXD[7:0], bit 7 is the msb and bit 0 is the lsb).

15.6.1.1 Media-Independent Interface (MII)

This section describes the media-independent interface (MII) intended to be used between the PHYs and the TSEC. Figure 15-110 shows the basic components of the MII including the signals required to establish TSEC module connection with a PHY.



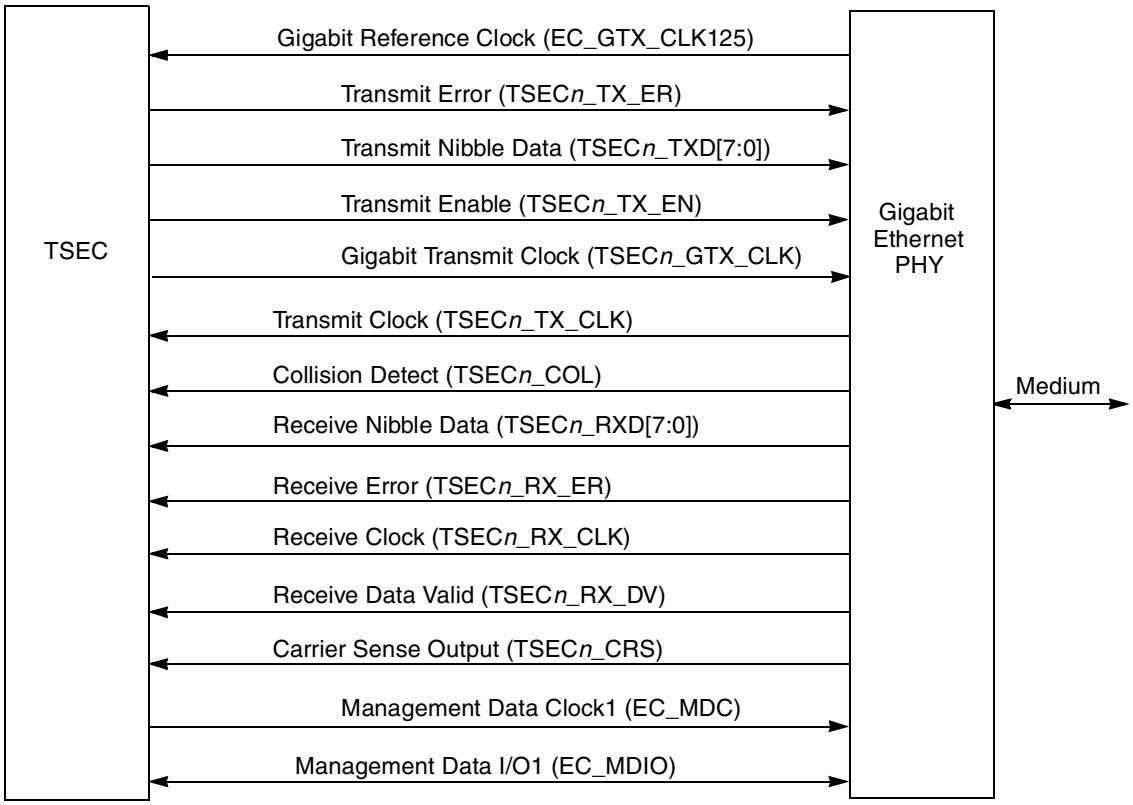
[1] The management signals (EC_MDC and EC_MDIO) are common to all of the Ethernet controllers' connections in the system, assuming that each PHY has a different management address.

Figure 15-109. TSEC-MII Connection

An MII interface has 18 signals (including the EC_MDC and EC_MDIO signals), as defined by the IEEE 802.3u standard, for connecting to an Ethernet PHY.

15.6.1.2 Gigabit Media-Independent Interface (GMII)

This section describes the gigabit media-independent interface (GMII) intended to be used between the PHYs and the TSEC. Figure 15-111 shows the basic components of the GMII including the signals required to establish the TSEC module connection with a PHY.



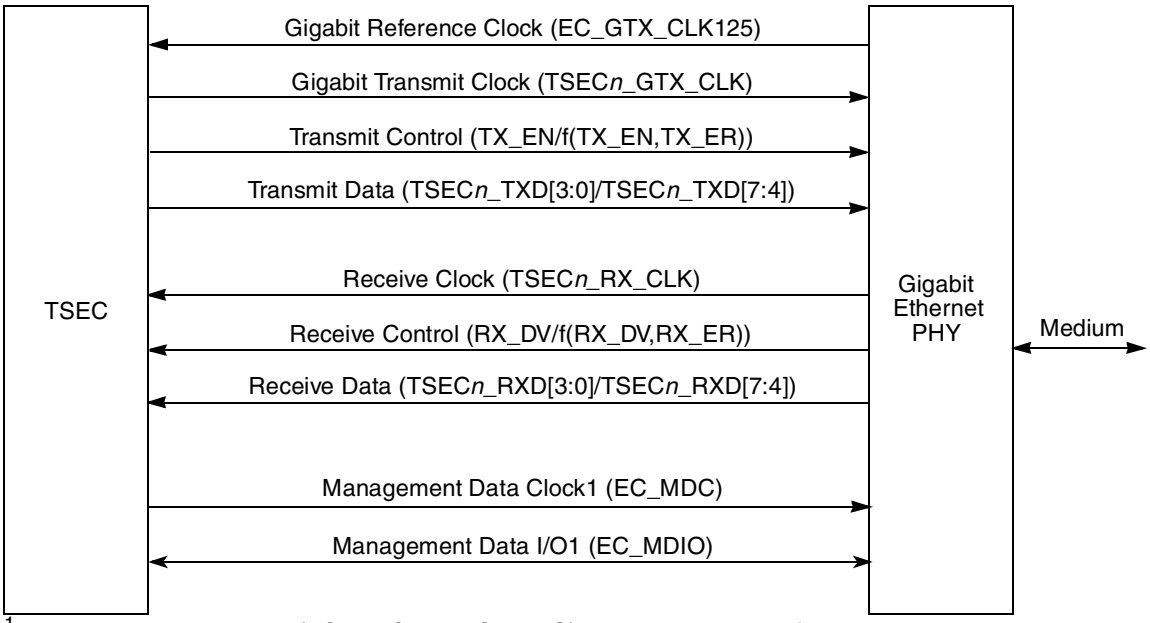
[1] The management signals (EC_MDC and EC_MDIO) are common to all of the Ethernet controllers' connections in the system, assuming that each PHY has a different management address.

Figure 15-110. TSEC-GMII Connection

A GMII interface has 28 signals (TSEC*n*_GTX_CLK + EC_GTX_CLK125 included), as defined by the IEEE 802.3u standard, for connecting to an Ethernet PHY.

15.6.1.3 Reduced Gigabit Media-Independent Interface (RGMII)

This section describes the reduced gigabit media-independent interface (RGMII) intended to be used between the PHYs and the GMII MAC. The RGMII is an alternative to the IEEE802.3u MII, the IEEE802.3z GMII, and the TBI. The RGMII reduces the number of signals required to interconnect the MAC and the PHY from a maximum of 28 signals (GMII) to 15 signals (EC_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. Figure 15-112 shows 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.

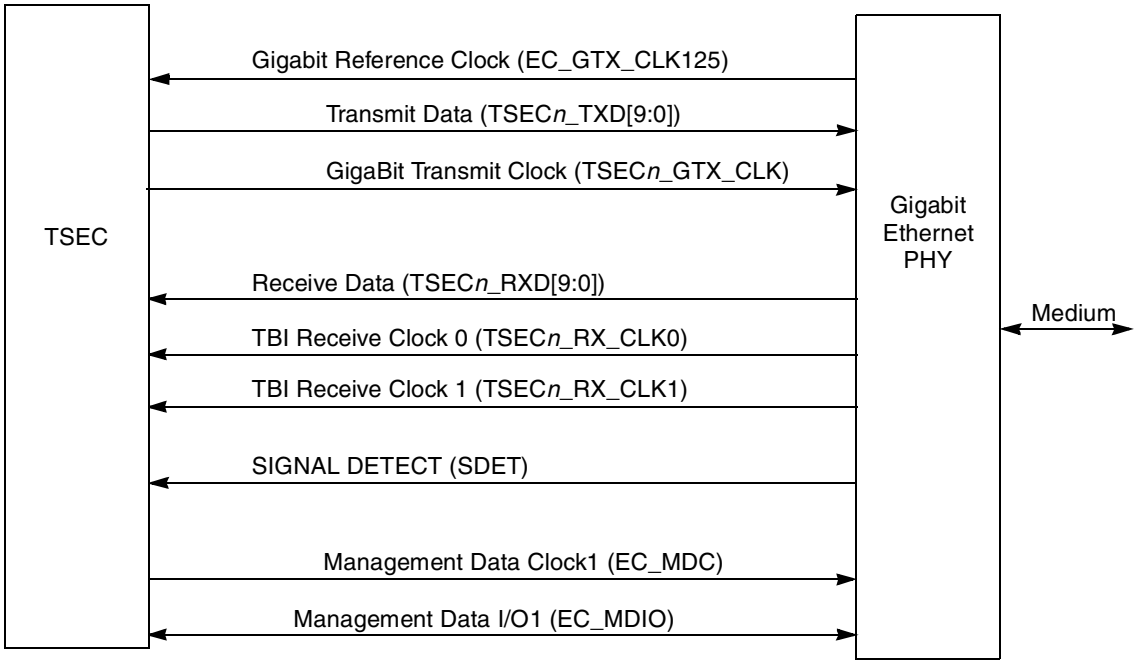


[1] The management signals (EC_MDC and EC_MDIO) are common to all of the gigabit Ethernet controllers module connections in the system, assuming that each PHY has a different management address.

Figure 15-111. TSEC-RGMII Connection

15.6.1.4 Ten-Bit Interface (TBI)

This section describes the ten-bit interface (TBI) intended to be used between the PHYs and the TSEC to implement a standard SerDes interface for optical-fiber devices in 1000BASE-SX/LX applications. Figure 15-113 shows the basic components of the TBI including the signals required to establish TSEC module connection with a PHY. RBC0 and RBC1 are differential 62.5-MHz receive clocks.



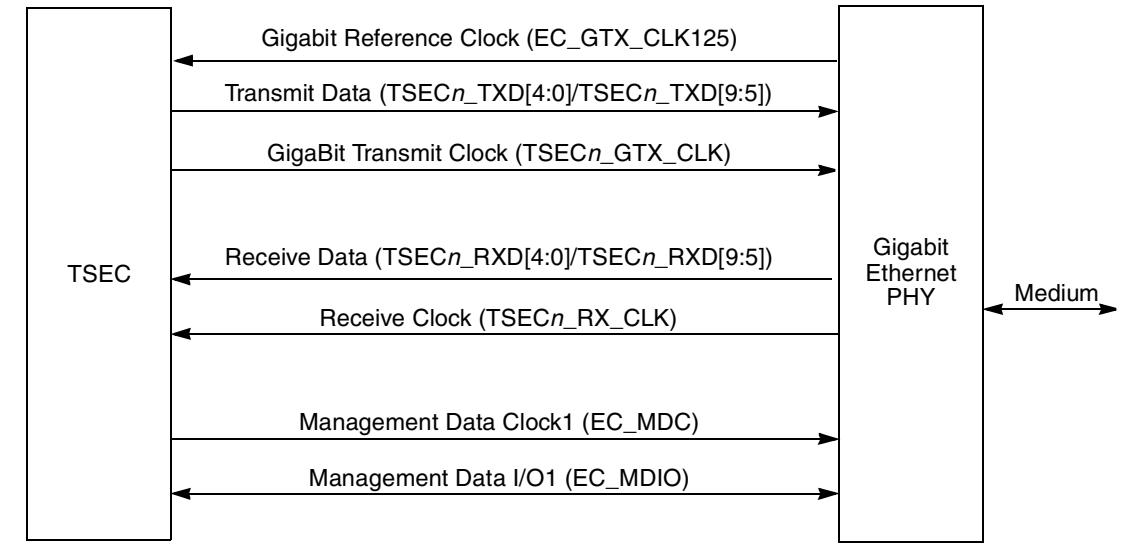
[1] The management signals (EC_MDC and EC_MDIO) are common to all of the Ethernet controllers' connections in the system, assuming that each PHY has a different management address.

Figure 15-112. TSEC-TBI Connection

A TBI interface has 27 signals (EC_GTX_CLK125 included) for connecting to an Ethernet PHY, as defined by IEEE 802.3z GMII and TBI standards.

15.6.1.5 Reduced Ten-Bit Interface (RTBI)

This section describes the reduced ten-bit interface (RTBI) intended to be used between the PHYs and the TSEC to implement a reduced signal count version of a SerDes interface for optical-fiber devices in 1000BASE-SX/LX applications. Figure 15-114 shows the basic components of the RTBI including the signals required to establish TSEC module connection with a PHY.



[1] The management signals (EC_MDC and EC_MDIO) are common to all of the Ethernet controllers' connections in the system, assuming that each PHY has a different management address.

Figure 15-113. TSEC-RTBI Connection

A RTBI interface has 15 signals (EC_GTX_CLK125 included), as defined by the RGMII specification Version 1.2a 9/22/00, and is intended to be an alternative to the IEEE 802.3u MII, the IEEE 802.3z GMII and the TBI standard for connecting to an Ethernet PHY.

Table 15-109 describes the signal multiplexing for GMII, MII and TBI interfaces.

Table 15-109. GMII, MII, and TBI Signal Multiplexing

TSEC Signals Frequency [MHz] 125 Voltage[V] 3.3/2.5			GMII Interface Frequency [MHz] 125 Voltage[V] 3.3			MII Interface Frequency [MHz] 25 Voltage[V] 3.3			TBI Interface Frequency [MHz] 62.5 Voltage[V] 3.3		
Signals (TSEC*n*_)	**I/O**	**No. of Signals**	**Signals (TSEC*n*_)**	**I/O**	**No. of Signals**	**Signals (TSEC*n*_)**	**I/O**	**No. of Signals**	**Signals (TSEC*n*_)**	**I/O**	**No. of Signals**
GTX_CLK	O	1	GTX_CLK	O	1				GTX_CLK	O	1
TX_CLK	I	1	TX_CLK	I	1	TX_CLK	I	1	RX_CLK1	I	1
TxD[0]	O	1	TxD[0]	O	1	TxD[0]	O	1	TCG[0]	O	1
TxD[1]	O	1	TxD[1]	O	1	TxD[1]	O	1	TCG[1]	O	1
TxD[2]	O	1	TxD[2]	O	1	TxD[2]	O	1	TCG[2]	O	1
TxD[3]	O	1	TxD[3]	O	1	TxD[3]	O	1	TCG[3]	O	1
TxD[4]	O	1	TxD[4]	O	1				TCG[4]	O	1
TxD[5]	O	1	TxD[5]	O	1				TCG[5]	O	1
TxD[6]	O	1	TxD[6]	O	1				TCG[6]	O	1
TxD[7]	O	1	TxD[7]	O	1				TCG[7]	O	1
TX_EN	O	1	TX_EN	O	1	TX_EN	O	1	TCG[8]	O	1

Table 15-109. GMII, MII, and TBI Signal Multiplexing (continued)

TSEC Signals Frequency [MHz] 125 Voltage[V] 3.3/2.5			GMII Interface Frequency [MHz] 125 Voltage[V] 3.3			MII Interface Frequency [MHz] 25 Voltage[V] 3.3			TBI Interface Frequency [MHz] 62.5 Voltage[V] 3.3		
TX_ER	O	1	TX_ER	O	1	TX_ER	O	1	TCG[9]	O	1
RX_CLK	I	1	RX_CLK	I	1	RX_CLK	I	1	RX_CLK0	I	1
RxD[0]	I	1	RxD[0]	I	1	RxD[0]	I	1	RCG[0]	I	1
RxD[1]	I	1	RxD[1]	I	1	RxD[1]	I	1	RCG[1]	I	1
RxD[2]	I	1	RxD[2]	I	1	RxD[2]	I	1	RCG[2]	I	1
RxD[3]	I	1	RxD[3]	I	1	RxD[3]	I	1	RCG[3]	I	1
RxD[4]	I	1	RxD[4]	I	1				RCG[4]	I	1
RxD[5]	I	1	RxD[5]	I	1				RCG[5]	I	1
RxD[6]	I	1	RxD[6]	I	1				RCG[6]	I	1
RxD[7]	I	1	RxD[7]	I	1				RCG[7]	I	1
RX_DV	I	1	RX_DV	I	1	RX_DV	I	1	RCG[8]	I	1
RX_ER	I	1	RX_ER	I	1	RX_ER	I	1	RCG[9]	I	1
COL	I	1				COL	I	1			
CRS	I	1				CRS	I	1	SDET	I	1
Sum		25	**Sum**		23	**Sum**		16	**Sum**		24

Table 15-110 describes the signal multiplexing for RGMII and RTBI interfaces.

Table 15-110. RGMII and RTBI Signal Multiplexing

TSEC Signals Frequency [MHz] 125 Voltage[V] 3.3/2.5			RGMII Interface Frequency [MHz] 125 Voltage[V] 2.5			RTBI Interface Frequency [MHz] 62.5 Voltage[V] 2.5		
Signals (TSEC*n*_)	**I/O**	**No. of Signals**	**Signals (TSEC*n*_)**	**I/O**	**No. of Signals**	**Signals (TSEC*n*_)**	**I/O**	**No. of Signals**
GTX_CLK	O	1	GTX_CLK	O	1	GTX_CLK	O	1
TX_CLK	I	1						
TxD[0]	O	1	TxD[0]/TxD[4]	O	1	TCG[0]/TCG[5]	O	1
TxD[1]	O	1	TxD[1]/TxD[5]	O	1	TCG[1]/TCG[6]	O	1
TxD[2]	O	1	TxD[2]/TxD[6]	O	1	TCG[2]/TCG[7]	O	1
TxD[3]	O	1	TxD[3]/TxD[7]	O	1	TCG[3]/TCG[8]	O	1
TxD[4]	O	1						
TxD[5]	O	1						

Table 15-110. RGMII and RTBI Signal Multiplexing (continued)

TSEC Signals Frequency [MHz] 125 Voltage[V] 3.3/2.5			RGMII Interface Frequency [MHz] 125 Voltage[V] 2.5			RTBI Interface Frequency [MHz] 62.5 Voltage[V] 2.5		
Signals (TSEC*n*_)	**I/O**	**No. of Signals**	**Signals (TSEC*n*_)**	**I/O**	**No. of Signals**	**Signals (TSEC*n*_)**	**I/O**	**No. of Signals**
TxD[6]	O	1						
TxD[7]	O	1						
TX_EN	O	1	TX_CTL (TX_EN/TX_ERR)	O	1	TCG[4]/TCG[9]	O	1
TX_ER	O	1						
RX_CLK	I	1	RX_CLK	I	1	RX_CLK	I	1
RxD[0]	I	1	RxD[0]/RxD[4]	I	1	RCG[0]/RCG[5]	I	1
RxD[1]	I	1	RxD[1]/RxD[5]	I	1	RCG[1]/RCG[6]	I	1
RxD[2]	I	1	RxD[2]/RxD[6]	I	1	RCG[2]/RCG[7]	I	1
RxD[3]	I	1	RxD[3]/RxD[7]	I	1	RCG[3]/RCG[8]	I	1
RxD[4]	I	1						
RxD[5]	I	1						
RxD[6]	I	1						
RxD[7]	I	1						
RX_DV	I	1	RX_CTL (RX_DV/RX_ERR)	I	1	RCG[4]/RCG[9]	I	1
RX_ER	I	1						
COL	I	1						
CRS	I	1						
Sum		25	**Sum**		12	**Sum**		12

Table 15-111 describes the signals shared by all interfaces.

Table 15-111. Shared Signals

Signals	I/O	# of Signals	Function
EC_MDIO	I/O	1	Management interface I/O
EC_MDC	O	1	Management interface clock
EC_GTX_CLK125	I	1	Reference clock
Sum		3	

15.6.2 Gigabit Ethernet Channel Operation

This section describes the operation of the TSEC. First, the software initialization sequence is described. Next, the software (Ethernet driver) interface for transmitting and receiving frames is reviewed. Address recognition and hash table algorithm features are also discussed. The section concludes with interrupt handling, inter-packet gap time, and loop back descriptions.

15.6.2.1 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 TSEC.

15.6.2.1.1 Hardware Controlled Initialization

A hard reset occurs when the system powers up. All TSEC registers and control logic are reset to their default states after a hard reset has occurred.

15.6.2.1.2 User Initialization

After the system has undergone a hard reset, software must initialize certain basic TSEC 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. The module memory map in Table 15-3 lists all the TSEC registers. Table 15-112 describes the minimum steps for register initialization.

Table 15-112. Steps of Minimum Register Initialization

Description
1. Set, then clear MACCFG1 [Soft_Reset]
2. Initialize MACCFG2
3. Initialize MAC station address
4. Set up the PHY using the MII Mgmt Interface
5. Configure the TBI control to TBI or GMII

Table 15-112. Steps of Minimum Register Initialization (continued)

Description
6. Clear IEVENT
7. Initialize IMASK
8. Initialize IADDR*n*
9. Initialize GADDR*n*
10. Initialize RCTRL
11. Initialize DMACTRL

After the registers are initialized, the user must execute the following steps in the order described below to bring the TSEC into a functional state (out of reset):

1. For the transmission of Ethernet frames, TxBDs must first be built in memory, linked together as a ring, and pointed to by the TBASE register. A minimum of two buffer descriptors per ring is required. Setting the ring to a size of one causes the same frame to be transmitted twice.
2. Likewise, for the reception of Ethernet frames, the receive queue must be ready, with its RxBD pointed to by the RBASE register. Both transmit and receive can be gracefully stopped after transmission and reception begins.
3. Write to MACCFG1 and set the appropriate bits. These need to include Rx_EN and Tx_EN. To enable flow control, Rx_Flow and Tx_Flow must also be set.
4. Clearing DMACTRL[GTS] triggers the transmission of frame data if the transmitter had been previously stopped. DMACTRL[GRS] must be cleared if the receiver had been previously stopped. See 15.5.3.1.7, “DMA Control Register (DMACTRL),” and Section 15.6.3.1, “Transmit Data Buffer Descriptor (TxBD),” for more information.

15.6.2.2 Soft Reset and Reconfiguring Procedure

Before issuing a soft reset to and/or reconfiguring the MAC with new parameters, 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 DMACTRL[GRS, GTS], then wait for both IEVENT[GRSC, GTSC] to be set 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 register. Likewise if a new set of Rx buffer descriptors are used, the RBASE register must be written with the new pointer.

Following is a procedure to gracefully reset and reconfigure the MAC:

1. Set DMACTRL[GTS].
2. Poll IEVENT[GTSC] until detected as set.
3. Clear MACCFG1[Rx_EN, Tx_EN].
4. Set DMACTRL[GRS].
5. Poll IEVENT[GRSC] until it is detected as set.

6. Set MACCFG1[Soft_Reset]
7. Clear MACCFG1[Soft_Reset]
8. Load TBASE with new TxBD pointer
9. Load RBASE with new RxBD pointer
10. Set up other MAC registers (MACCFG2, MAXFRM, etc.)
11. Set WWR, WOP, TOD bits in DMACTRL register
12. Clear THLT bit in TSTAT register and QHLT bit in RSTAT register by writing 1 to these bits.
13. Clear GRS/GTS bits in DMACTRL (do not change other bits)
14. Enable Tx_EN/Rx_EN in MACCFG1 register

15.6.2.3 Gigabit Ethernet Frame Transmission

The Ethernet transmitter requires little core intervention. After the software driver initializes the system, the TSEC begins to poll the first transmit buffer descriptor (TxBD) in the TxBD ring every 512 transmit clocks. If TxBD[R] is set, TSEC begins moving transmit buffer from memory to its Tx FIFO. The transmitter takes data from the Tx FIFO and transmits data to the MAC. The MAC transmits the data through the GMII interface to the physical media. The transmitter, once initialized, runs until the end-of-frame (EOF) condition is detected unless a collision within the collision window occurs (half-duplex mode) or an abort condition is encountered.

If the user has a frame ready to transmit, a transmit-on-demand function may be emulated while in polling mode by using the graceful-transmit-stop feature. First, clear the IMASK[GTSCEN] bit to mask the graceful-transmit-stop complete interrupt. Next set, then immediately clear the DMACTRL[GTS] bit. Clear the resulting IEVENT[GTSC] bit. Finally, the IMASK[GTSCEN] bit may be set once again.

There is one internal buffer for out-of-sequence flow control frames. While the TSEC is between frames, this buffer is polled. The buffer must contain the whole frame. Once the TSEC is in paused mode, the out-of-sequence buffer descriptor cannot be used to send another flow control frame because the MAC regards it as a regular TxBD.

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 improves bus usage and latency.

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 transmit block also 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 when a pause frame is received, 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 via 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 EN] is set

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.

To pause transmission or rearrange the transmit queue, set DMACTRL[GTS]. This can help in transmitting expedited data ahead of previously linked buffers or for error situations. If GTS is set, the TSEC transmitter performs a graceful transmit stop. The Ethernet controller stops immediately if no transmission is in progress or continues transmission until the current frame either finishes or terminates with an error. The IEVENT[GTSC] interrupt occurs once the graceful transmit stop operation is completed. After GTS is cleared, the TSEC resumes transmission with the next frame.

While the TSEC is in 10/100 Mbps mode it sends bytes least-significant nibble first and each nibble is sent lsb first. While it is in 1-Gbps mode it sends bytes lsb first.

15.6.2.4 Gigabit Ethernet Frame Reception

The TSEC Ethernet receiver is designed to work with little core intervention and can perform 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. If TSEC*n*_RX_DV is asserted and TSEC*n*_COL remains negated, the MAC strips a valid preamble/SFD (start of frame delimiter) header and begins processing the frame. If a valid header is not found, the frame is ignored.

If the receiver detects the first bytes of a frame, the TSEC controller begins to perform the frame recognition function through destination address (DA) recognition (See Section 15.6.2.6, “Frame Recognition,” for additional information.). Based on this match the frame can be accepted or rejected. Once accepted, the TSEC processes the frame based on user-defined attributes.

The receiver can also filter frames based on physical (individual), group (multicast), and broadcast addresses. Because Ethernet receive frame data is not written to memory until the internal frame recognition algorithm is complete, CSB 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 the queue. If RxBD is not being used by software (RxBD[E] is set), the TSEC starts transferring the incoming frame. RxBD[F] is set for the first RxBD used for any particular receive frame.

If the buffer is filled, the TSEC controller 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 (MRBL). The smallest valid value is 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 a frame-too-long condition.

Receive frames are not truncated if 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 TSEC 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 signalled after the graceful receive stop operation is completed. While in this mode the user can then clear IEVENT[GRSC] and can write to registers that are accessible to both the user and the TSEC hardware without fear of conflict. After DMACTRL[GRS] is cleared, the TSEC 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.

15.6.2.5 RMON Support

TSEC automatically gathers network statistics required for RMON without needing to receive all addresses. The RMON MIB group 1, RMON MIB group 2, RMON MIB group 3, RMON MIB group 9, RMON MIB 2, and the 802.3 Ethernet MIB are supported.

For RMON statistics and their corresponding counters see the memory map.

15.6.2.6 Frame Recognition

The Ethernet controller performs frame recognition using destination address (DA) recognition. A frame can be rejected or accepted based on the outcome.

15.6.2.6.1 Destination Address Recognition

The Ethernet controller can also perform the frame filtering using the traditional destination address (DA) recognition methods.

Figure 15-115 is a flowchart for address recognition on received frames that is used to explain the concept. In the actual implementation most of the decision points shown in the figure actually occur simultaneously.

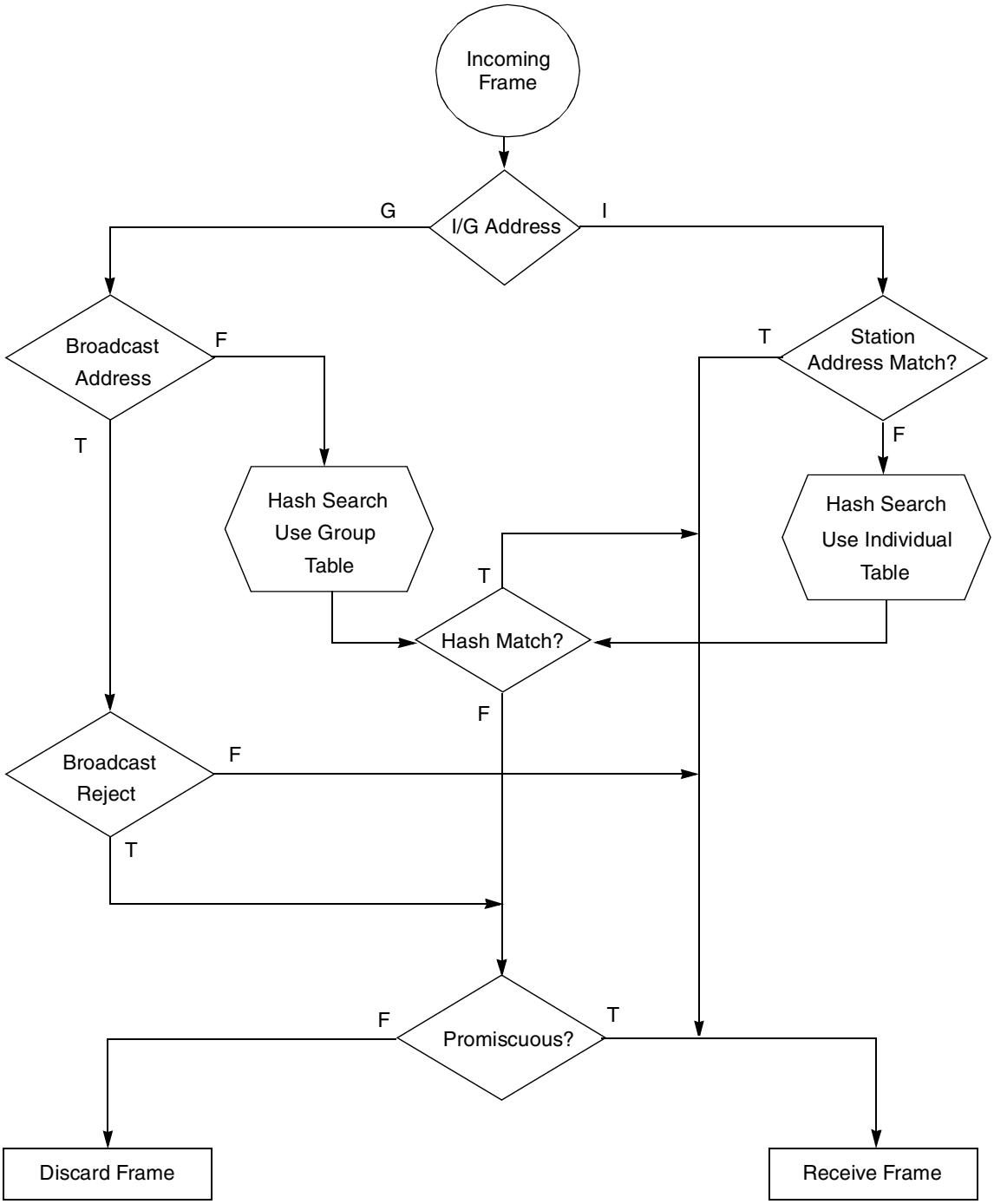


Figure 15-114. Ethernet Address Recognition Flowchart

The Ethernet controller compares the destination address field of the received frame with the physical address the user programs in the station address registers (MACSTNADDR1 and MACSTNADDR2). If the DA does not match the station address, then the controller performs address recognition on multiple individual addresses using the IADDR*n* hash table. The user must write zeros to the hash in order to avoid a hash match, and ones to station address in order to avoid individual address match, or the user can turn on promiscuous mode. (See section Section 15.5.3.4.1, “Receive Control Register (RCTRL).”)

In the group type of address recognition, the Ethernet controller determines whether the group address is a broadcast address. If it is a broadcast, and broadcast addresses are enabled, the frame is accepted. If the group address is not a broadcast address, the user can perform address recognition on multiple group addresses using the GADDR*n* hash table. In promiscuous mode, the Ethernet controller receives all of the incoming frames regardless of their address.

15.6.2.6.2 Hash Table Algorithm

The hash table process used in the individual and group hash filtering operates as follows. The Ethernet controller maps any 48-bit destination address into one of 256 bins, represented by the 256 bits in GADDR0–7 or IADDR0–7. The eight high-order bits of a cyclic redundancy check (CRC) checksum are used to index into the has table. The high-order three bits of this 8-bit field are used to select one of the eight registers in either the individual or group hash table. The low-order five bits select a bit within the 32-bit register. A value of 0 in the high-order three bits selects IADDR0/GADDR0.

The same process is used if the Ethernet controller receives a frame. If the CRC checksum selects a bit that is set in the group/individual hash table, the frame is accepted. If 32 group addresses are stored in the hash table and random group addresses are received, the hash table prevents roughly 224/256 (87.5%) of the group address frames from reaching memory. Software must further filter those that reach memory to determine if they contain the correct addresses.

Better performance is achieved by using the group and individual hash tables in combination. For instance, if 32 group and 32 physical addresses are stored in their respective hash tables, because 87.5% of all group addresses and 87.5% of all individual address are rejected, then 87.5% of all frames are prevented from reaching memory.

The effectiveness of the hash table declines as the number of addresses increases. For instance, as the number of addresses stored in the 256-bin hash table increases, the vast majority of the hash table bits are set, preventing only a small fraction of frames from reaching memory.

15.6.2.7 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 15-113 lists the flow-control frame structure.

Table 15-113. Flow Control Frame Structure

Size [Octets]	Description	Value	Comment
7	Preamble		
1	SFD		Start frame delimiter
6	Destination address	01-80C2-00-00-01	Multicast address reserved for use in MAC frames
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 sent first.
2	Extended MAC parameter		Extended pause control parameter as defined by the PTV[PTE] field. The most-significant octet is sent 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. During this pause, only a control frame can be sent (TCTRL[TFC_PAUSE] is set). Normal transmission resumes after the pause timer stops counting. If another pause-control frame is received during the pause, the period changes to the new value received.

15.6.2.8 Interrupt Handling

The following describes what usually occurs within a TSEC 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.
- Process the TxBDs to reuse them if the IEVENT[TXB or TXF] were set. If the transmit speed is fast or the interrupt delay is long, more than one transmit buffer may have been sent by the TSEC; 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. See Table 15-114.
- Obtain data from the RxBD if IEVENT[RXC,RXB or RXF] is set. If the receive speed is fast or the interrupt delay is long, the TSEC may have received more than one RxBD; thus, it is important to check more than just one RxBD during interrupt handling. Typically, all RxBDs in the interrupt handler are processed until one is found with E set. Because the TSEC pre-fetches BDs, the BD table must be big enough so that there is always another empty BD to pre-fetch. See Table 15-115.

- Clear any set halt bits in TSTAT and RSTAT registers, or DMACTRL[GTS] and DMACTRL[GRS].
- Continue normal execution.

Table 15-114. Non-Error Transmit Interrupts

Interrupt	Description	Action taken by TSEC
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.	Programmable “write with response” TxBD to memory before setting IEVENT[TXB].
TXF	Transmit frame: A frame 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 15-115. Non-Error Receive Interrupts

Interrupt	Description	Action taken by TSEC
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.	Programmable write response RxBD to memory before setting IEVENT[RXB].
RXF	Receive frame: A frame was received and the last receive buffer descriptor (RxBD) of that frame was updated.	Programmable write response to memory on the last RxBD before setting IEVENT[RXF].

15.6.2.8.1 Interrupt Coalescing

Interrupt coalescing offers the user the ability to contour the behavior of the TSEC with regard to frame interrupts. Separate but identical mechanisms exist for handling both transmitted frames and received frames. While interrupt coalescing is enabled a transmit or receive frame interrupt (resulting from the interrupt bit (I) of the buffer descriptor in question and appropriately-enabled by the IMASK register) is raised either when a counter threshold-defined number of frames is received/transmitted or the timer threshold-defined period of time has lapsed, whichever occurs first.

15.6.2.8.2 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 (ICFCT) in the appropriate interrupt coalescing configuration register (RXIC or TXIC). The frame threshold field may be assigned a value between 1 and 255. A value of 0 results in boundedly undefined behavior. Note that a value of 1 functionally defeats the advantages of interrupt coalescing since the frame threshold is reached with each frame received or transmitted. Once the number of frames transmitted or received reaches the threshold limit, an interrupt is raised (if the interrupt bit of the frame buffer descriptor(s) is set and if appropriately-enabled in the IMASK register), the threshold counter is reset, and then continues counting frames while the interrupt is active. The threshold counter is also reset if an interrupt is raised subject to the corresponding threshold timer.

15.6.2.8.3 Interrupt Coalescing 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 (if the interrupt bit of the frame buffer descriptor(s) is set and if enabled in the IMASK register). 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 TSEC interface clocks. That is, one timer threshold unit is 26.5 µs, 2.56 µs, or 512 ns, corresponding to interface modes 10 Mbps, 100 Mbps, or 1 Gbps, respectively.

After transmitting a frame (with TXBD[I] set, causing IEVENT[TXF] to be set), the transmit interrupt coalescing threshold timer begins counting. When the timer threshold has been reached an interrupt is raised if the interrupt bit of the buffer descriptor is set and IMASK[TXFEN] is set.

After receiving a frame (with RXBD[I] set, causing IEVENT[RXF] to be set), the receive interrupt coalescing threshold timer begins counting. When the timer threshold has been reached an interrupt is raised if the interrupt bit of the buffer descriptor is set and IMASK[RXFEN] is set.

The interrupt coalescing timer thresholds (transmit and receive, operating independently) may be values ranging from 1 to 65535. A value of 0 is illegal and results in behavior identical to that when interrupt coalescing is disabled (corresponding RXIC[ICEN] or TXIC[ICEN] is cleared). Table 15-116 specifies the range of possible timing thresholds subject to the interface frequency and the value of RXIC[ICTT] or TXIC[ICTT].

Table 15-116. Interrupt Coalescing Timing Threshold Ranges

TSEC Interface Format and Frequency	Interrupt Coalescing Threshold Time	
	Minimum (ICTT=1)	Maximum (ICTT=65535)
10BaseT at 2.5 MHz	25.6 µs	1.678 s
100BaseT at 25 MHz	2.56 µs	167.8 ms
1000BaseT at 125 MHz	512 ns	33.55 ms

The transmit or receive timer threshold counter is reset when a corresponding interrupt is raised and begins counting again upon deassertion of that interrupt and once IEVENT[TXF or RXF] is set.

15.6.2.9 Inter-Packet Gap Time

If a station must transmit, it waits until the LAN becomes silent for a specified period (inter-packet gap). 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 and then begins retransmission on the LAN. This process is called a retry. If the packet is not successfully sent within a specified number of retries, an error is indicated.

The minimum inter-packet gap time for back-to-back transmission is 96 serial clocks. The receiver receives back-to-back packets with this minimum spacing. In addition, after waiting a required number of clocks (based on the back-off algorithm), the transmitter waits for carrier sense to be negated before retransmitting the packet. Retransmission begins 36 serial clocks after carrier sense is negated for at least 60 serial clocks.

15.6.2.10 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.

15.6.2.11 Error-Handling Procedure

The Ethernet controller reports frame reception and transmission error conditions using the channel BDs, the error counters, and the IEVENT register.

Programming note: When the TSEC encounters a halt condition (TSTAT[THLT] is set), it stops processing the frame at the current TxBD. The TSEC relies on the user to manage the buffer descriptor pointer, TBPTR, or the buffer descriptor queue before resuming transmissions. Once the TSEC resumes, it fetches the TxBD pointed to by TBPTR.

Transmission errors are described in Table 15-117.

Table 15-117. Transmission Errors

Error	Response
Transmitter underrun	The controller sends 32 bits that ensure a CRC error, terminates buffer transmission, sets TxBD[UN], closes the buffer, sets 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, sets IEVENT[CRL/XDA] and IEVENT[TXE]. Transmission resumes after TSTAT[THLT] is cleared (and DMACTRL[GTS] is cleared).
Excessive defer abort	The controller terminates buffer transmission, sets TxBD[DEF], closes the buffer, sets IEVENT[CRC/XDA], and IEVENT[TXE]. Transmission resumes after TSTAT[THLT] is cleared.
Late collision	The controller terminates buffer transmission, sets TxBD[LC], closes the buffer, sets IEVENT[LC] and IEVENT[TXE]. The controller resumes transmission after TSTAT[THLT] is cleared (and DMACTRL[GTS] is cleared).

Table 15-117. Transmission Errors (continued)

Error	Response
Memory Read Error	A CSB error occurred during a DMA transaction. The controller sets IEVENT[EBERR], DMA stops sending data to the FIFO which causes an underrun error but IEVENT[XFUN] is not set. The TSTAT[THLT] is set. Transmits are continued once TSTAT[THLT] is cleared.
Babbling Transmit Error	A frame is transmitted which exceeds the MAC's maximum frame length and MACCFG2[HFE]= 0. The controller sets IEVENT[BABT] and continues without interruption. TxBD[TXTRUNC] is set in the last TxBD (TxBD[L] is set) of the frame.

Reception errors are described in Table 15-118.

Table 15-118. Reception Errors

Error	Description
Overrun error	The Ethernet controller maintains an internal FIFO buffer for receiving data. If a receiver FIFO buffer overrun occurs, the controller sets RxBD[OV], sets RxBD[L], closes the buffer, 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]. In addition, the RSTAT[QHLT] bit is set. The halted queue resumes reception once the user clears the RSTAT[QHLT] bit.
Non-octet error (dribbling bits)	The Ethernet controller handles a nibble of dribbling bits if the receive frame terminates as non-octet aligned and it checks the CRC of the frame on the last octet boundary. If there is a CRC error, the frame non-octet aligned (RxBD[NO]) error is reported, IEVENT[RXF] is set, and the alignment error counter increments. The TSEC relies on the statistics collector block to increment the receive alignment error counter (RALN). If there is no CRC error, no error is reported.
CRC error	If a CRC error occurs, the controller sets RxBD[CR], closes the buffer, and sets IEVENT[RXF]. This TSEC relies on the statistics collector block to record the event. After receiving a frame with a CRC error, the receiver then enters hunt mode.
Memory read error	A CSB error occurred during a DMA transaction. The controller sets IEVENT[EBERR] and discards the frame. In addition the RSTAT[QHLT] bit is set. The halted queue resumes reception once the RSTAT[QHLT] bit is cleared.
Babbling receive error	A frame is received that exceeds the MAC's maximum frame length. The controller sets IEVENT[BABR] and continues

15.6.3 Buffer Descriptors

The TSEC buffer descriptor (BD) is modeled after the MPC8260 Fast Ethernet controller BD for ease of reuse. Drawing from the MPC8260 FEC BD programming model, the TSEC descriptor base registers point to the beginning of BD rings. The 8-byte data BD format is similar to the MPC8260 BD model.

Data buffers are used in the transmission and reception of Ethernet frames (see Figure 15-116). Data BDs encapsulate all information necessary for the TSEC 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 two buffer descriptors per ring are required. This applies to both the transmit and the receive descriptor rings. Software also must have the data pointer

pointing to memory. 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 TSEC and the software driver.

Because there is no next BD pointer in the transmit/receive BD (see Figure 15-117), all BDs must reside sequentially in memory. The TSEC 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 TSEC to loop back to the beginning of the BD chain. Software must initialize TBASE and RBASE that point to the beginning transmit and receive BDs for TSEC.

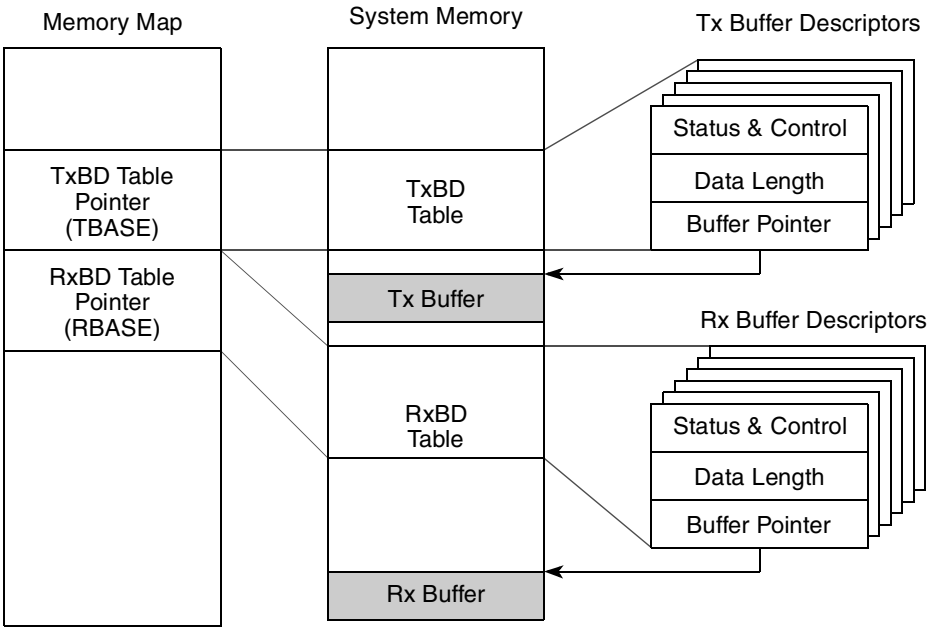


Figure 15-115. Example of TSEC Memory Structure for BD

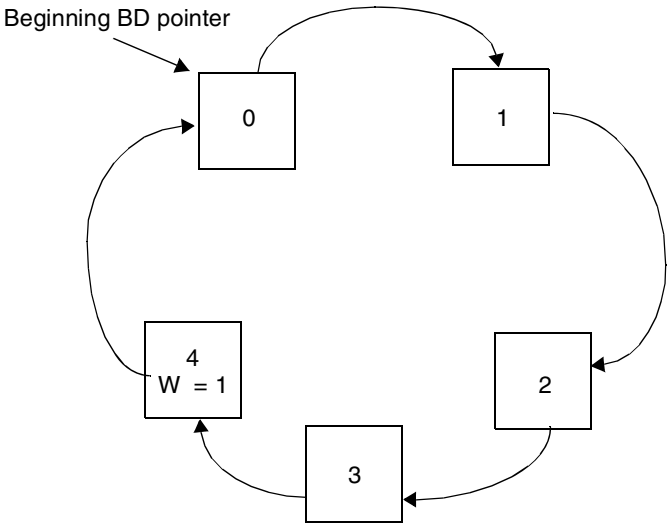


Figure 15-116. Buffer Descriptor Ring

15.6.3.1 Transmit Data Buffer Descriptor (TxBD)

Data is presented to the TSEC for transmission by arranging it in memory buffers referenced by the TxBDs. In the TxBD the user initializes the R, PAD/CRC, W, L, and TC bits and the length (in bytes) in the first word, and the buffer pointer in the second word.

The TSEC 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. Figure 15-118 shows the TxBD.

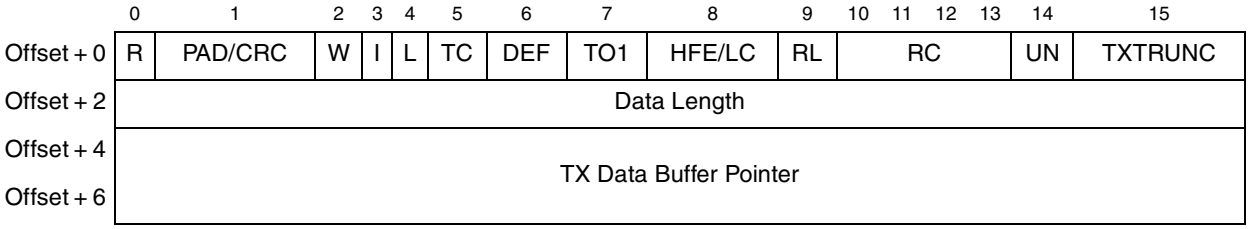


Figure 15-117. Transmit Buffer Descriptor

The TxBD fields are detailed in Table 15-119.

Table 15-119. Transmit Data Buffer Descriptor (TxBD) Field Descriptions

Offset	Bits	Name	Description
Offset + 0	0	R	Ready. Written by TSEC 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 TSEC 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.
Offset + 0	1	PAD/ CRC	PAD/CRC. Padding and CRC attachment for frames. (Valid only while it is set in the first BD and MACCFG2[PAD/CRC] is cleared.) If MACCFG2[PAD/CRC] is set, this bit is ignored. 0 Do not add padding to short frames. No CRC is appended unless TxBD[TC] is set. 1 Add PAD/CRCs 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, TSEC PADs always up to the IEEE minimum frame length of 64 bytes. CRC is always appended to frames.
Offset + 0	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.
Offset + 0	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).
Offset + 0	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 15-119. Transmit Data Buffer Descriptor (TxBD) Field Descriptions (continued)

Offset	Bits	Name	Description
Offset + 0	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] is cleared and MACCFG2[CRC EN] is cleared.) If MACCFG2[PAD/CRC] is set or MACCFG2[CRC EN] is set, this bit is ignored. 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.
Offset + 0	6	DEF	Defer indication. Hardware updates this bit if an excessive defer condition occurs. 0 This frame was not deferred. 1 If HAFDUP[EXCESS_DEFER]=1, this frame did not have a collision before it was sent but it was sent late because of deferring. If HAFDUP[EXCESS_DEFER]=0, this frame was aborted and not sent.
Offset + 0	7	TO1	Transmit software ownership. This read/write bit may be utilized by software, as necessary. Its state does not affect the hardware nor is it affected by the hardware.
Offset + 0	8	HFE/LC	Huge frame enable/Late collision. Huge frame enable. Written by user. Valid only while it is set in first BD and the 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 register. 1 Do not truncate the transmit frame. Late collision. Written by TSEC. 0 No late collision. 1 A collision occurred after 64 bytes are sent. The TSEC terminates the transmission and updates LC.
Offset + 0	9	RL	Retransmission limit. Written by TSEC. 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 TSEC terminates the transmission and updates RL.
Offset + 0	10–13	RC	Retry count. Written by TSEC. 0 The frame is sent correctly the first time or if RL is set then the retry limit has been reached x One or more attempts were 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.
Offset + 0	14	UN	Underrun. Written by TSEC. 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. The TSEC terminates the transmission and updates UN.
Offset + 0	15	TXTRUNC	TX truncation. Set in the last TxBD (TxBD[L] is set) when IEVENT[BABT] occurs for the frame.
Offset + 2	0–15	Data Length	Data length is the number of octets the TSEC transmits from this BD's data buffer. It is never modified by the TSEC. This field must be greater than zero.
Offset + 4	0–31	TX Data Buffer Pointer	The transmit buffer pointer contains the address of the associated data buffer. There are no alignment requirements for this address.

15.6.3.2 Receive Buffer Descriptor (RxBD)

In the RxBD the user initializes the E, I, and W bits in the first word and the pointer in second word. If the data buffer is used, the TSEC modifies the E, L, F, M, BC, MC, LG, NO, SH, 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, SH, CR, OV, and TR bits in the first word of the buffer descriptor are only modified by the TSEC if the L bit is set. The first word of the RxBD contains control and status bits. Its format is detailed in Figure 15-119 below.

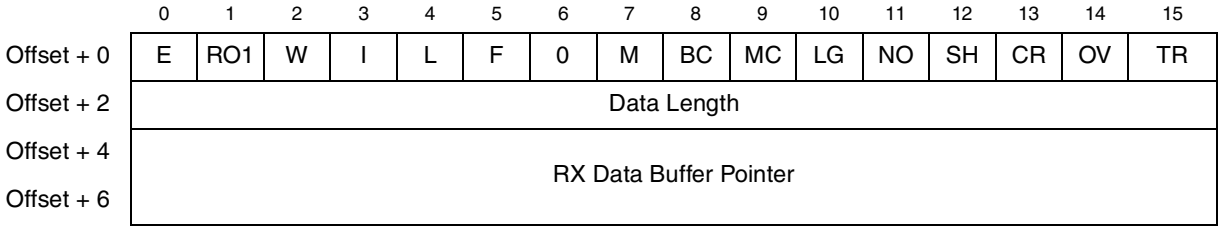


Figure 15-118. Receive Buffer Descriptor

Table 15-120 describes the fields of the RxBD.

Table 15-120. Receive Buffer Descriptor Field Descriptions

Offset	Bits	Name	Description
Offset + 0	0	E	Empty. Written by TSEC (when cleared) and by 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.
Offset + 0	1	RO1	Receive software ownership bit. Reserved for use by software. This read/write bit is not modified by hardware, nor does its value affect hardware.
Offset + 0	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.
Offset + 0	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] is set or IMASK[RXFEN] is set). 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).
Offset + 0	4	L	Last in frame. Written by TSEC. 0 The buffer is not the last in a frame. 1 The buffer is the last in a frame.
Offset + 0	5	F	First in frame. Written by TSEC. 0 The buffer is not the first in a frame. 1 The buffer is the first in a frame.
Offset + 0	6	—	Reserved

Table 15-120. Receive Buffer Descriptor Field Descriptions (continued)

Offset	Bits	Name	Description
Offset + 0	7	M	Miss. Written by TSEC. (This bit is valid only if the L-bit is set and TSEC is in promiscuous mode.) This bit is set by the TSEC 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.
Offset + 0	8	BC	Broadcast. Written by TSEC. (Only valid if L is set.) Is set if the DA is broadcast (FF-FF-FF-FF-FF-FF).
Offset + 0	9	MC	Multicast. Written by TSEC. (Only valid if L is set.) Is set if the DA is multicast and not BC.
Offset + 0	10	LG	Rx frame length violation. Written by TSEC. (Only valid if L is set.) A frame length greater than maximum frame length was recognized. 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.
Offset + 0	11	NO	Rx non-octet aligned frame. Written by TSEC. (Only valid if L is set.) A frame that contained a number of bits not divisible by eight was received.
Offset + 0	12	SH	Short frame. Written by TSEC. (only valid if L is set.) A frame length that was less than the minimum length defined for this channel (MINFLR) was recognized, provided RCTRL[RSF] is set.
Offset + 0	13	CR	Rx CRC error. Written by TSEC. (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.
Offset + 0	14	OV	Overrun. Written by TSEC. (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, SH, and CR lose their normal meaning and are zero.
Offset + 0	15	TR	Truncation. Written by TSEC. (Only valid if L is set.) Is set if the receive frame is truncated. This can happen if a frame length greater than maximum frame length was received and the 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.
Offset + 2	0–15	Data Length	Data length. Written by TSEC. Data length is the number of octets written by the TSEC into this BD's data buffer if L is cleared (the value is equal to MRBL), or the length of the frame including CRC, if L is set.
Offset + 4	0–31	Rx Data Buffer Pointer	Receive buffer pointer. Written by user. The receive buffer pointer, which always points to the first location of the associated data buffer, must be 64-byte aligned. The buffer must reside in memory external to the TSEC.

15.7 Initialization/Application Information

15.7.1 Interface Mode Configuration

This section describes how to configure the TSEC in different supported interface modes. These include MII, GMII, TBI, RGMII, and RTBI. The pinout, the data registers that must be initialized, as well as speed selection options are described. The ECNTRL[TBIM] and ECNTRL[RPM] bits are written, assuming the part was not pin-configured at initialization to the correct mode.

15.7.1.1 MII Interface Mode

Table 15-121 describes the signal configurations required for MII interface mode.

Table 15-121. MII Interface Mode Signal Configuration

TSEC Signals Frequency [MHz] 125 Voltage[V] 3.3/2.5			MII Interface Frequency [MHz] 25 Voltage[V] 3.3		
Signals	**I/O**	**No. of Signals**	**Signals**	**I/O**	**No. of Signals**
GTX_CLK	O	1	Leave Unconnected	O	
TX_CLK	I	1	TX_CLK	I	1
TxD[0]	O	1	TxD[0]	O	1
TxD[1]	O	1	TxD[1]	O	1
TxD[2]	O	1	TxD[2]	O	1
TxD[3]	O	1	TxD[3]	O	1
TxD[4]	O	1	Leave Unconnected	O	
TxD[5]	O	1	Leave Unconnected	O	
TxD[6]	O	1	Leave Unconnected	O	
TxD[7]	O	1	Leave Unconnected	O	
TX_EN	O	1	TX_EN	O	1
TX_ER	O	1	TX_ER	O	1
RX_CLK	I	1	RX_CLK	I	1
RxD[0]	I	1	RxD[0]	I	1
RxD[1]	I	1	RxD[1]	I	1
RxD[2]	I	1	RxD[2]	I	1
RxD[3]	I	1	RxD[3]	I	1
RxD[4]	I	1	not used	I	
RxD[5]	I	1	not used	I	
RxD[6]	I	1	not used	I	

Table 15-121. MII Interface Mode Signal Configuration (continued)

TSEC Signals Frequency [MHz] 125 Voltage[V] 3.3/2.5			MII Interface Frequency [MHz] 25 Voltage[V] 3.3		
Signals	**I/O**	**No. of Signals**	**Signals**	**I/O**	**No. of Signals**
RxD[7]	I	1	not used	I	
RX_DV	I	1	RX_DV	I	1
RX_ER	I	1	RX_ER	I	1
COL	I	1	COL	I	1
CRS	I	1	CRS	I	1
Sum		25	**Sum**		16

Table 15-122 describes the shared signals of the MII interface.

Table 15-122. Shared MII Signals

TSEC Signals	I/O	No. of Signals	MII Signals	I/O	No. of Signals	Function
EC_MDIO	I/O	1	EC_MDIO	I/O	1	Management interface I/O
EC_MDC	O	1	EC_MDC	O	1	Management interface Clock
EC_GTX_CLK125	I	1	not used	I	0	Reference Clock
Sum		3	**Sum**		2	

Table 15-123 describes the register initializations required to reconfigure the PHY to MII mode following initial auto-negotiation.

Table 15-123. MII Mode Register Initialization Steps

Select G/MII mode at Reset Configuration Word (RCWH[TSECxM])
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)

Table 15-123. MII Mode Register Initialization Steps (continued)

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. (Note that PHY configuration register addresses are not necessarily consistent in all PHYs.)
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. (Note that PHY configuration register addresses are not necessarily consistent in all PHYs.)
Assign a Physical address to the TBI so as to not conflict with the external PHY Physical address, TBIPA[0000_0000_0000_0000_0000_0000_0000_0101] Set to 05, for example. (Note that PHY configuration register addresses are not necessarily consistent in all PHYs.)
Reset the management interface. MIIMCFG[1000_0000_0000_0000_0000_0000_0000_0000]
Setup the MII Mgmt clock speed, MIIMCFG[0000_0000_0000_0000_000_0000_0000_0101] set source clock divide by 14, for example, to insure that EC_MDC clock speed is approximately 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 TSEC 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.
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.

Table 15-123. MII Mode Register Initialization Steps (continued)

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_000_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[1111_1111_1111_1111_1111_1111_1111_1111]
Initialize IMASK, (Optional) IMASK[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize IADDR*n,* (Optional) IADDR*n*[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize GADDR*n,* (Optional) GADDR*n*[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 TBASE, TBASE[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000]
Initialize (Empty) Receive Descriptor ring and fill with empty buffers. Initialize RBASE, RBASE[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000]
Enable Rx and Tx, MACCFG1[0000_0000_0000_0000_0000_0000_0000_0101]

15.7.1.2 GMII Interface Mode

Table 15-124 describes the signal configurations required for GMII interface mode.

Table 15-124. GMII Interface Mode Signal Configuration

TSEC SIGNALS Frequency [MHz] 125 Voltage[V] 3.3/2.5			GMII INTERFACE Frequency [MHz] 125 Voltage[V] 3.3		
Signals	**I/O**	**No. of Signals**	**Signals**	**I/O**	**No. of Signals**
GTX_CLK	O	1	GTX_CLK	O	1
TX_CLK	I	1			
TxD[0]	O	1	TxD[0]	O	1
TxD[1]	O	1	TxD[1]	O	1
TxD[2]	O	1	TxD[2]	O	1
TxD[3]	O	1	TxD[3]	O	1
TxD[4]	O	1	TxD[4]	O	1
TxD[5]	O	1	TxD[5]	O	1
TxD[6]	O	1	TxD[6]	O	1
TxD[7]	O	1	TxD[7]	O	1
TX_EN	O	1	TX_EN	O	1
TX_ER	O	1	TX_ER	O	1
RX_CLK	I	1	RX_CLK	I	1
RxD[0]	I	1	RxD[0]	I	1
RxD[1]	I	1	RxD[1]	I	1
RxD[2]	I	1	RxD[2]	I	1
RxD[3]	I	1	RxD[3]	I	1
RxD[4]	I	1	RxD[4]	I	1
RxD[5]	I	1	RxD[5]	I	1
RxD[6]	I	1	RxD[6]	I	1
RxD[7]	I	1	RxD[7]	I	1
RX_DV	I	1	RX_DV	I	1
RX_ER	I	1	RX_ER	I	1
COL	I	1			
CRS	I	1			
Sum		25	**Sum**		22

Table 15-125 describes the shared signals of the GMII interface.

Table 15-125. Shared GMII Signals

TSEC Signals	I/O	No. of Signals	GMII Signals	I/O	No. of Signals	Function
EC_MDIO	I/O	1	EC_MDIO	I/O	1	Management interface I/O
EC_MDC	O	1	EC_MDC	O	1	Management interface Clock
EC_GTX_CLK125	I	1	EC_GTX_CLK125	I	1	Reference Clock
Sum		3	**Sum**		3	

Table 15-126 describes the register initializations required to reconfigure the PHY to GMII mode following initial auto-negotiation.

Table 15-126. GMII Mode Register Initialization Steps

Select G/MII mode at Reset Configuration Word (RCWH[TSECxM]).
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 GMII, Full duplex operation. Set I/F Mode bit. MACCFG2[0000_0000_0000_0000_0111_0010_0000_0101] (This example has Full Duplex = 1, 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. (Note that PHY configuration register addresses are not necessarily consistent in all PHYs.)
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. (Note that PHY configuration register addresses are not necessarily consistent in all PHYs.)
Assign a Physical address to the TBI so as to not conflict with the external PHY Physical address, TBIPA[0000_0000_0000_0000_0000_0000_0000_0101] Set to 05, for example. (Note that PHY configuration register addresses are not necessarily consistent in all PHYs.)
Reset the management interface, MIIMCFG[1000_0000_0000_0000_0000_0000_0000_0000]
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 EC_MDC clock speed is approximately 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 TSEC MII Mgmt bus is idle.

Table 15-126. GMII Mode Register Initialization Steps (continued)

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. Write 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
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_000u_00u1_0100_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_000_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]

Table 15-126. GMII Mode Register Initialization Steps (continued)

Initialize IMASK, (Optional) IMASK[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize IADDR*n*, (Optional) IADDR*n*[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize GADDR*n*, (Optional) GADDR*n*[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 TBASE, TBASE[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000]
Initialize (Empty) Receive Descriptor ring and fill with empty buffers. Initialize RBASE, RBASE[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000]
Enable Rx and Tx, MACCFG1[0000_0000_0000_0000_0000_0000_0000_0101]

15.7.1.3 TBI Interface Mode

Table 15-127 describes the signal configurations required for TBI interface mode.

Table 15-127. TBI Interface Mode Signal Configuration

TSEC Signals Frequency [MHz] 125 Voltage[V] 3.3/2.5			TBI Interface Frequency [MHz] 62.5 Voltage[V] 3.3		
Signals	**I/O**	**No. of Signals**	**Signals**	**I/O**	**No. of Signals**
GTX_CLK	O	1	GTX_CLK	O	1
TX_CLK	I	1	RX_CLK1	I	1
TxD[0]	O	1	TCG[0]	O	1
TxD[1]	O	1	TCG[1]	O	1
TxD[2]	O	1	TCG[2]	O	1
TxD[3]	O	1	TCG[3]	O	1
TxD[4]	O	1	TCG[4]	O	1
TxD[5]	O	1	TCG[5]	O	1
TxD[6]	O	1	TCG[6]	O	1
TxD[7]	O	1	TCG[7]	O	1
TX_EN	O	1	TCG[8]	O	1

Table 15-127. TBI Interface Mode Signal Configuration (continued)

TSEC Signals Frequency [MHz] 125 Voltage[V] 3.3/2.5			TBI Interface Frequency [MHz] 62.5 Voltage[V] 3.3		
Signals	**I/O**	**No. of Signals**	**Signals**	**I/O**	**No. of Signals**
TX_ER	O	1	TCG[9]	O	1
RX_CLK	I	1	RX_CLK0	I	1
RxD[0]	I	1	RCG[0]	I	1
RxD[1]	I	1	RCG[1]	I	1
RxD[2]	I	1	RCG[2]	I	1
RxD[3]	I	1	RCG[3]	I	1
RxD[4]	I	1	RCG[4]	I	1
RxD[5]	I	1	RCG[5]	I	1
RxD[6]	I	1	RCG[6]	I	1
RxD[7]	I	1	RCG[7]	I	1
RX_DV	I	1	RCG[8]	I	1
RX_ER	I	1	RCG[9]	I	1
COL	I	1	SDET	I	1
CRS	I	1			
Sum		25	**Sum**		24

Table 15-128 describes the shared signals for the TBI interface.

Table 15-128. Shared TBI Signals

TSEC Signals	I/O	No. of Signals	TBI Signals	I/O	No. of Signals	Function
EC_MDIO	I/O	1	EC_MDIO	I/O	1	Management interface I/O
EC_MDC	O	1	EC_MDC	O	1	Management interface Clock
EC_GTX_CLK125	I	1	EC_GTX_CLK125	I	1	Reference Clock
Sum		3	**Sum**		3	

Table 15-129 describes the register initializations required to reconfigure the PHY to TBI mode following initial auto-negotiation.

Table 15-129. TBI Mode Register Initialization Steps

Select TBI mode at Reset Configuration Word (RCWH[TSECxM]
Set Soft_Reset, MACCFG1[1000_0000_0000_0000_0000_0000_0000_0000]
Clear Soft_Reset, MACCFG1[0000_0000_0000_0000_0000_0000_0000_0101]
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 Statistics Enable = 1)
Initialize MAC Station Address MACSTNADDR2[0110_0000_0000_0010_0000_0000_0000_0000] to 02608C:876543, for example. (Note that PHY configuration register addresses are not necessarily consistent in all PHYs.)
Initialize MAC Station Address MACSTNADDR1[0100_0011_0110_0101_1000_0111_1000_1100] to 02608C:876543, for example. (Note that PHY configuration register addresses are not necessarily consistent in all PHYs.)
Assign a Physical address to the TBI, TBIPA[0000_0000_0000_0000_0000_0000_0001_0000] set to 16, for example. (Note that PHY configuration register addresses are not necessarily consistent in all PHYs.)
Setup the MII Mgmt clock speed, MIIMCFG[0000_0000_0000_0000_0000_0000_0000_0111] Set source clock divide by 28, for example, to insure that EC_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 TSEC MII Mgmt bus is idle.
Set up the MII Mgmt for a read cycle to TBI Control register (write the TBI address and Register address), MIIMADD[0000_0000_0000_0000_0001_0000_0000_0000] The TBI Control register is at offset address 0x0 from TBIPA.
Perform an MII Mgmt read cycle to verify state of TBI Control Register(0ptional) Clear MIIMCOM[Read Cycle] Set MIIMCOM[Read Cycle] (Uses the TBI address and Register address placed in MIIMADD register), When MIIMIND[BUSY]=0, read the MIIMSTAT and look for AN Enable and other bit information.
Set up the MII Mgmt for a write cycle to TBI's AN Advertisement register (write the PHY address and Register address), MIIMADD[0000_0000_0000_0000_0001_0000_0000_0100] The AN Advertisement register is at offset address 0x04 from the TBI's address. (in this case 0x10)

Table 15-129. TBI Mode Register Initialization Steps (continued)

Perform an MII Mgmt write cycle to TBI. Writing to MII Mgmt Control with 16-bit data intended for TBI's AN Advertisement register, MIIMCON[0000_0000_0000_0000_0000_0001_1010_0000] This advertises to the Link Partner that the TBI supports PAUSE and Full Duplex mode and does not support Half Duplex mode.
Check to see if MII Mgmt write is complete. 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 TBI's Control register (write the PHY address and Register address), MIIMADD[0000_0000_0000_0000_0001_0000_0000_0000] the Control register is at offset address 0x00 from the TBI's address. (in this case 0x10)
Perform an MII Mgmt write cycle to TBI. Writing to MII Mgmt Control with 16-bit data intended for TBI's Control register, MIIMCON[0000_0000_0000_0000_0001_0010_0000_0000] This enables the TBI to start Auto-Negotiations using the configuration set in the AN Advertisement register.
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.
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_0001_0000_0000_0001] The Phy Status control register is at address 0x1 and in this case the PHY Address is 0x10.
Perform an MII Mgmt read cycle of Status Register. 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 ---> [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)
Perform an MII Mgmt read cycle of AN Expansion Register. Setup MIIMADD[0000_0000_0000_0000_0001_0000_0000_0110] Clear MIIMCOM[Read Cycle] Set MIIMCOM[Read Cycle] (Uses the PHY address (0x10) 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 ---> [0000_0000_0000_0000_0000_0000_0000_0110]
Perform an MII Mgmt read cycle of AN Link Partner Base Page Ability Register. (Optional) Setup MIIMADD[0000_0000_0000_0000_0001_0000_0000_0101] Clear MIIMCOM[Read Cycle] Set MIIMCOM[Read Cycle] (Uses the PHY address (0x10) 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 2 and 3 (Remote Fault) and bits 9 and 10. (Half and Full Duplex) MII Mgmt AN Link Partner Base Page Ability ---> [0000_0000_0000_0000_0000_000x_x110_0000]

Table 15-129. TBI Mode Register Initialization Steps (continued)

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 IADDR*n* (Optional) IADDR*n*[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize GADDR*n* (Optional) GADDR*n*[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 TBASE, TBASE[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000]
Initialize (Empty) Receive Descriptor ring and fill with empty buffers Initialize RBASE, RBASE[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000]
Enable Rx and Tx, MACCFG1[0000_0000_0000_0000_0000_0000_0000_0101]

15.7.1.4 RGMII Interface Mode

Table 15-130 shows the signals configurations required for RGMII interface mode.

Table 15-130. RGMII Interface Mode Signal Configuration

TSEC SIGNALS Frequency [MHz] 125 Voltage[V] 3.3/2.5			RGMII INRTERFACE Frequency [MHz] 125 Voltage[V] 2.5		
Signals	**I/O**	**No. of Signals**	**Signals**	**I/O**	**No. of Signals**
GTX_CLK	O	1	GTX_CLK	O	1
TX_CLK	I	1			
TxD[0]	O	1	TxD[0]/TxD[4]	O	1
TxD[1]	O	1	TxD[1]/TxD[5]	O	1
TxD[2]	O	1	TxD[2]/TxD[6]	O	1
TxD[3]	O	1	TxD[3]/TxD[7]	O	1
TxD[4]	O	1			
TxD[5]	O	1			
TxD[6]	O	1			

Table 15-130. RGMII Interface Mode Signal Configuration (continued)

TSEC SIGNALS Frequency [MHz] 125 Voltage[V] 3.3/2.5			RGMII INRTERFACE Frequency [MHz] 125 Voltage[V] 2.5		
Signals	**I/O**	**No. of Signals**	**Signals**	**I/O**	**No. of Signals**
TxD[7]	O	1			
TX_EN	O	1	TX_CTL (TX_EN/TX_ERR)	O	1
TX_ER	O	1			
RX_CLK	I	1	RX_CLK	I	1
RxD[0]	I	1	RxD[0]/RxD[4]	I	1
RxD[1]	I	1	RxD[1]/RxD[5]	I	1
RxD[2]	I	1	RxD[2]/RxD[6]	I	1
RxD[3]	I	1	RxD[3]/RxD[7]	I	1
RxD[4]	I	1			
RxD[5]	I	1			
RxD[6]	I	1			
RxD[7]	I	1			
RX_DV	I	1	RX_CTL (RX_DV/RX_ERR)	I	1
RX_ER	I	1			
COL	I	1			
CRS	I	1			
Sum		25	**Sum**		12

Table 15-131 describes the shared signals for the RGMII interface.

Table 15-131. Shared RGMII Signals

TSEC Signals	I/O	No. of Signals	RGMII Signals	I/O	No. of Signals	Function
EC_MDIO	I/O	1	EC_MDIO	I/O	1	Management interface I/O
EC_MDC	O	1	EC_MDC	O	1	Management interface Clock
EC_GTX_CLK125	I	1	EC_GTX_CLK125	I	1	Reference Clock
Sum		3	**Sum**		3	

Table 15-132 describes the register initializations required to reconfigure the PHY to RGMII mode following initial auto-negotiation.

Table 15-132. RGMII Mode Register Initialization Steps

Select RGMII mode at Reset Configuration Word (RCWH[TSECxM]
Set Soft_Reset, MACCFG1[1000_0000_0000_0000_0000_0000_0000_0000]
Clear Soft_Reset, MACCFG1[0000_0000_0000_0000_0000_0000_0000_0101]
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 10-Mbps mode, Statistics Enable = 1)
Initialize MAC Station Address, MACSTNADDR2[0110_0000_0000_0010_0000_0000_0000_0000] to 02608C:876543, for example. (Note that PHY configuration register addresses are not necessarily consistent in all PHYs.)
Initialize MAC Station Address, MACSTNADDR1[0100_0011_0110_0101_1000_0111_1000_1100] to 02608C:876543, for example. (Note that PHY configuration register addresses are not necessarily consistent in all PHYs.)
Assign a Physical address to the TBI, TBIPA[0000_0000_0000_0000_0000_0000_0001_0000] set to 16, for example. (Note that PHY configuration register addresses are not necessarily consistent in all PHYs.)
Setup the MII Mgmt clock speed, MIIMCFG[0000_0000_0000_0000_000_0000_0000_0111] Set source clock divide by 28, for example, to insure that EC_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 TSEC 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 15-132. RGMII Mode Register Initialization Steps (continued)

Set up the MII Mgmt for a write cycle to the external PHY Control register (write the PHY address and Register address), MIIMADD[0000_0000_0000_0000_0001_0001_0000_0000] The Control register is at offset address 0x00 from the external PHY address. (in this case 0x11)
Perform an MII Mgmt write cycle to the external PHY. 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 start Auto-Negotiations using the configuration set in the AN Advertisement register.
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.
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.
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 ---> [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)
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 ---> [0000_0000_0000_0000_0000_0000_0000_0110]
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 ---> [0000_0000_0000_0000_0000_000x_x110_0000]
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 IADDR*n* (Optional) IADDR*n*[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize GADDR*n* (Optional) GADDR*n*[0000_0000_0000_0000_0000_0000_0000_0000]

Table 15-132. 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 TBASE, TBASE[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000]
Initialize (Empty) Receive Descriptor ring and fill with empty buffers Initialize RBASE, RBASE[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000]
Enable Rx and Tx, MACCFG1[0000_0000_0000_0000_0000_0000_0000_0101]

15.7.1.5 RTBI Interface Mode

Table 15-133 describes the signal configurations required for RTBI interface mode.

Table 15-133. RTBI Interlace Mode Signal Configuration

TSEC Signals Frequency [MHz] 125 Voltage[V] 3.3/2.5			RTBI Interface Frequency [MHz] 62.5 Voltage[V] 2.5		
Signals	**I/O**	**No. of Signals**	**Signals**	**I/O**	**No. of Signals**
GTX_CLK	O	1	GTX_CLK	O	1
TX_CLK	I	1			
TxD[0]	O	1	TCG[0]/TCG[5]	O	1
TxD[1]	O	1	TCG[1]/TCG[6]	O	1
TxD[2]	O	1	TCG[2]/TCG[7]	O	1
TxD[3]	O	1	TCG[3]/TCG[8]	O	1
TxD[4]	O	1			
TxD[5]	O	1			
TxD[6]	O	1			
TxD[7]	O	1			
TX_EN	O	1	TCG[4]/TCG[9]	O	1
TX_ER	O	1			
RX_CLK	I	1	RX_CLK	I	1
RxD[0]	I	1	RCG[0]/RCG[5]	I	1
RxD[1]	I	1	RCG[1]/RCG[6]	I	1

Table 15-133. RTBI Interlace Mode Signal Configuration (continued)

TSEC Signals Frequency [MHz] 125 Voltage[V] 3.3/2.5			RTBI Interface Frequency [MHz] 62.5 Voltage[V] 2.5		
Signals	**I/O**	**No. of Signals**	**Signals**	**I/O**	**No. of Signals**
RxD[2]	I	1	RCG[2]/RCG[7]	I	1
RxD[3]	I	1	RCG[3]/RCG[8]	I	1
RxD[4]	I	1			
RxD[5]	I	1			
RxD[6]	I	1			
RxD[7]	I	1			
RX_DV	I	1	RCG[4]/RCG[9]	I	1
RX_ER	I	1			
COL	I	1			
CRS	I	1		I	
Sum		25	**Sum**		12

Table 15-134 describes the shared signals for the RTBI interface.

Table 15-134. Shared RTBI Signals

TSEC Signals	I/O	No. of Signals	RTBI Signals	I/O	No. of Signals	Function
EC_MDIO	I/O	1	EC_MDIO	I/O	1	Management interface I/O
EC_MDC	O	1	EC_MDC	O	1	Management interface Clock
EC_GTX_CLK125	I	1	EC_GTX_CLK125	I	1	Reference Clock
Sum		3	**Sum**		3	

Table 15-135 describes the register initializations required to reconfigure the PHY to RTBI mode following initial auto-negotiation.

Table 15-135. RTBI Mode Register Initialization Steps

Select RTBI mode at Reset Configuration Word (RCWH[TSECxM].
Set Soft_Reset, MACCFG1[1000_0000_0000_0000_0000_0000_0000_0000]
Clear Soft_Reset, MACCFG1[0000_0000_0000_0000_0000_0000_0000_0101]

Table 15-135. RTBI Mode Register Initialization Steps (continued)

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 Statistics Enable = 1)
Initialize MAC Station Address, MACSTNADDR2[0110_0000_0000_0010_0000_0000_0000_0000] to 02608C:876543, for example. (Note that PHY configuration register addresses are not necessarily consistent in all PHYs.)
Initialize MAC Station Address, MACSTNADDR1[0100_0011_0110_0101_1000_0111_1000_1100] to 02608C:876543, for example. (Note that PHY configuration register addresses are not necessarily consistent in all PHYs.)
Assign a Physical address to the TBI, TBIPA[0000_0000_0000_0000_0000_0000_0001_0000] set to 16, for example. (Note that PHY configuration register addresses are not necessarily consistent in all PHYs.)
Setup the MII Mgmt clock speed, MIIMCFG[0000_0000_0000_0000_0000_0000_0000_0111] Set source clock divide by 28, for example, to insure that EC_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 TSEC MII Mgmt bus is idle.
Set up the MII Mgmt for a read cycle to TBI Control register (write the TBI's address and Register address), MIIMADD[0000_0000_0000_0000_0001_0000_0000_0000] The TBI Control register is at offset address 0x0 from TBIPA.
Perform an MII Mgmt read cycle to verify state of TBI Control Register(0ptional) Clear MIIMCOM[Read Cycle] Set MIIMCOM[Read Cycle] (Uses the TBI address and Register address placed in MIIMADD register), When MIIMIND[BUSY]=0, read the MIIMSTAT and look for AN Enable and other bit information.
Set up the MII Mgmt for a write cycle to TBI's AN Advertisement register (write the PHY address and Register address), MIIMADD[0000_0000_0000_0000_0001_0000_0000_0100] The AN Advertisement register is at offset address 0x04 from the TBI's address. (in this case 0x10)
Perform an MII Mgmt write cycle to TBI. Write to MII Mgmt Control with 16-bit data intended for TBI's AN Advertisement register, MIIMCON[0000_0000_0000_0000_0000_0001_1010_0000] This advertises to the Link Partner that the TBI supports PAUSE and Full Duplex mode and does not support Half Duplex mode.
Check to see if MII Mgmt write is complete. 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 15-135. RTBI Mode Register Initialization Steps (continued)

Set up the MII Mgmt for a write cycle to TBI's Control register (write the PHY address and Register address), MIIMADD[0000_0000_0000_0000_0001_0000_0000_0000] The Control register is at offset address 0x00 from the TBI's address. (in this case 0x10)
Perform an MII Mgmt write cycle to TBI. Writing to MII Mgmt Control with 16-bit data intended for TBI's Control register, MIIMCON[0000_0000_0000_0000_0001_0010_0000_0000] This enables the TBI to start Auto-Negotiations using the configuration set in the AN Advertisement register.
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.
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_0001_0000_0000_0001] The Phy Status control register is at address 0x1 and in this case the PHY Address is 0x10.
Perform an MII Mgmt read cycle of Status Register. 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 ---> [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)
Perform an MII Mgmt read cycle of AN Expansion Register. Setup MIIMADD[0000_0000_0000_0000_0001_0000_0000_0110] Clear MIIMCOM[Read Cycle] Set MIIMCOM[Read Cycle] (Uses the PHY address (0x10) 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 ---> [0000_0000_0000_0000_0000_0000_0000_0110]
Perform an MII Mgmt read cycle of AN Link Partner Base Page Ability Register. (Optional) Setup MIIMADD[0000_0000_0000_0000_0001_0000_0000_0101] Clear MIIMCOM[Read Cycle] Set MIIMCOM[Read Cycle] (Uses the PHY address (0x10) 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 ---> [0000_0000_0000_0000_0000_000x_x110_0000]
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 IADDR*n* (Optional) IADDR*n*[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize GADDR*n* (Optional) GADDR*n*[0000_0000_0000_0000_0000_0000_0000_0000]

Table 15-135. RTBI 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 TBASE, TBASE[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000]
Initialize (Empty) Receive Descriptor ring and fill with empty buffers Initialize RBASE, RBASE[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000]
Enable Rx and Tx, MACCFG1[0000_0000_0000_0000_0000_0000_0000_0101]

Chapter 16
Universal Serial Bus Interface

This chapter describes the universal serial bus (USB) interface. 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 Specification, Revision 2.0*
- *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 Specification for Universal Serial Bus, Revision 1.0*
- *USB 2.0 Transceiver Macrocell Interface (UTMI) Specification, Version 1.05*

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*

16.1 Introduction

The MPC8349EA implements two USB modules, a multi-port host (MPH) module and a dual-role (DR) module. These modules may be connected to one or two external ports. Collectively the modules and external ports are called the USB interface. The USB interface is shown in Figure 16-1.

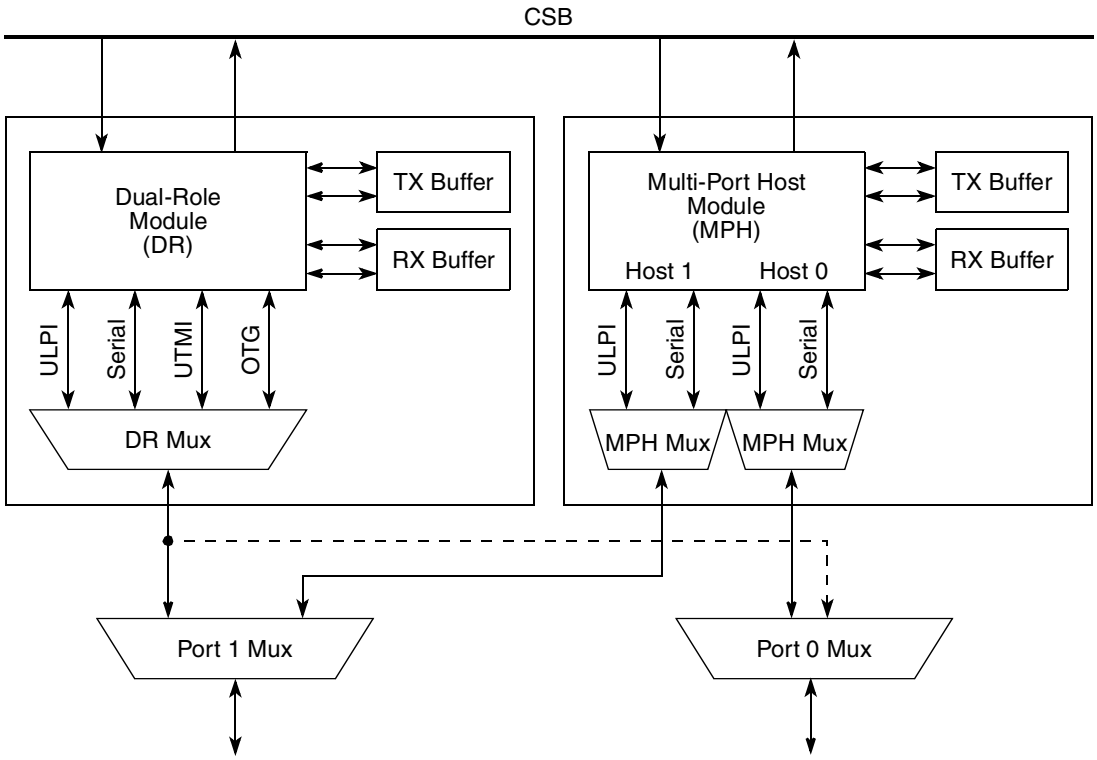


Figure 16-1. USB Interface Block Diagram

16.1.1 Overview

MPH and DR modules are USB 2.0-compliant serial interface engines for implementing a USB interface. The registers and data structures for both modules are based on the *Enhanced Host Controller Interface Specification for Universal Serial Bus* (EHCI) from Intel Corporation. The MPH module can act as a host controller for either one or two USB ports. The DR module can act as a host, as a device, or as an On-The-Go (OTG) negotiable host/device on the USB bus.

The DR module has the required signalling for USB transceiver macrocell interface (UTMI) or UTMI low pincount interface (ULPI) transceivers (PHYs) or for a full-speed (FS) or low-speed (LS) serial interface. Each host in the MPH module has the required signalling for either a ULPI PHY or a FS/LS serial interface.

Each module contains a chaining direct memory access (DMA) 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.

16.1.2 Features

The MPH module includes the following features:

- Complies with USB specification rev 2.0
- Supports operation as a standalone USB host controller
  - Supports USB root hub with one or two downstream facing ports
  - Supports enhanced host controller interface (EHCI)
- Supports high-speed (480 Mbps), full-speed (12 Mbps), and low-speed (1.5 Mbit/s) operations
- Supports external PHYs with ULPI and serial interfaces

The 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)
- USB device mode
- USB On-The-Go mode including host capability
- Supports high-speed (480 Mbps), full-speed (12 Mbps), and low-speed host (1.5 Mbit/s) operations
- Supports external PHYs with ULPI, UTMI, and serial interfaces
- Supports operation as a standalone USB device
  - Supports one upstream facing port
  - Supports six programmable, bidirectional USB endpoints
- Host mode supports direct connect of FS/LS devices

16.1.3 Modes of Operation

The MPH module provides the functionality of up to two USB 2.0 hosts. For the MPH module, each host can be configured to connect to either an ULPI PHY or a Serial FS/LS transceiver. However, if both host ports are in use, they must both be configured to the same PHY type. Any unused MPH host port must be suspended.

The DR module has three basic operating modes: host, device, and OTG. The DR module can be configured to use one of three different PHY interfaces: UTMI, ULPI, or FS/LS serial transceiver. Due to pin limitations, the UTMI interface can be used only in the device operating mode. Also note that LS operation is not supported by the DR module in device modes.

There are eight supported configurations for the USB modules and external ports. These configurations are summarized in Table 16-1.

Table 16-1. External Port Configurations

Config No.	External USB Port 1	External USB Port 0
DR Device		**MPH Host 0**
1	Host/Device/OTG, ULPI	Host, ULPI
2	Host/Device, Serial	Host, Serial
3	Host/Device/OTG, ULPI	Host, Serial
4	Host/Device, Serial	Host, ULPI
MPH Host 1		**MPH Host 0**
5	Host, ULPI	Host, ULPI
6	Host, Serial	Host, Serial
DR Device		
7	Device, UTMI	
DR OTG (Negotiable Host/Device)		
8	OTG, Serial OTG	

16.2 External Signals

This section contains detailed descriptions of the USB interface signals to reduce pin count; many of the signals for the PHY interfaces are muxed onto the same pin.

Table 16-2 describes the external signals functionality of the USB interface in all possible modes. Following sections will provide specific information per interface mode.

Note that the selection between MPH/DR per USB port is done by programming the system configuration registers SICRL and SICRH. See Section 5.3.2.5, “System I/O Configuration Register Low (SICRL),” and Section 5.3.2.6, “System I/O Configuration Register High (SICRH),” for more details. Note that the selection between ULPI/UTMI/Serial modes of operation per USB port is done by programming the USB MPH/DR Port Status and Control registers (PORTSC1/PORTSC2). Refer to Section 16.3.2.16, “Port Status and Control Registers (MPH—PORTSCn, DR—PORTSC1)” for more details.

Table 16-2. USB External Signals

Signal	I/O	Description
MPH1_D0_ENABLEN/ DR_D0_ENABLEN	I/O	ULPI—use as MPH1_DO or DR_D0 Serial—use as MPH1_ENABLEN or DR_ENABLEN UTMI—use as DR_D0
MPH1_D1_SER_TXD/ DR_D1_SER_TXD	I/O	ULPI—use as MPH1_D1 or DR_D1 Serial—use as MPH1_SER_TXD or DR_SER_TXD UTMI—use as DR_D1

Table 16-2. USB External Signals (continued)

Signal	I/O	Description
MPH1_D2_VMO_SE0/ DR_D2_VMO_SE0	I/O	ULPI—use as MPH1_D2 or DR_D2 Serial—use as MPH1_VMO_SE0 or DR_VMO_SE0 UTMI—use as DR_D2
MPH1_D3_SPEED/ DR_D3_SPEED	I/O	ULPI—use as MPH1_d3 or DR_D3 Serial—use as MPH1_SPEED or DR_SPEED UTMI—use as DR_D3
MPH1_D4_DP/ DR_D4_DP	I/O	ULPI—use as MPH1_D4 or DR_D4 Serial—use as MPH1_DP or DR_DP UTMI—use as DR_D4
MPH1_D5_DM/ DR_D5_DM	I/O	ULPI—use as MPH1_D5 or DR_D5 Serial—use as MPH1_DM or DR_DM UTMI—use as DR_D5
MPH1_D6_SER_RCV/ DR_D6_SER_RCV	I/O	ULPI—use as MPH1_D6 or DR_D6 Serial—use as MPH1_SER_RCV or DR_SER_RCV UTMI—use as DR_D6
MPH1_D7_DRVVBUS/ DR_D7_DRVVBUS	I/O	ULPI—use as MPH1_D7 or DR_D7 Serial—use as MPH1_DRVVBUS or DR_DRVVBUS UTMI—use as DR_D7
MPH1_NXT/ DR_SESS_VLD_NXT	I	ULPI—use as MPH1_NXT or DR_NXT Serial—use as DR_SESS_VLD (DR module-only) Serial OTG—use as DR_SESS_VLD UTMI—use as DR_SESS_VLD
MPH1_DIR_DPPULLUP/ DR_XCVR_SEL_DPPULLUP	I/O	ULPI—use as MPH1_DIR or DR_DIR Serial—use as MPH1_DPPULLUP or DR_DPPULLUP UTMI—use as DR_XCVR_SEL
MPH1_STP_SUSPEND/ DR_STP_SUSPEND	O	ULPI—use as MPH1_STP or DR_STP Serial—use as MPH1_SUSPEND or DR_SUSPEND UTMI—use as DR_SUSPEND
MPH1_PWRFAULT/ DR_RX_ERROR_PWRFAULT	I	ULPI—use as MPH1_PWRFAULT or DR_PWRFAULT Serial—use as MPH1_PWRFAULT or DR_PWRFAULT UTMI—use as DR_RX_ERROR
MPH1_PCTL0/ DR_TX_VALID_PCTL0	O	ULPI—use as MPH1_PCTL0 or DR_PCTL0 Serial—use as MPH1_PCTL0 or DR_PCTL0 UTMI—use as DR_TX_VALID
MPH1_PCTL1/ DR_TX_VALIDH_PCTL1	O	ULPI—use as MPH1_PCTL1 or DR_PCTL1 Serial—use as MPH1_PCTL1 or DR_PCTL1 UTMI—use as DR_TX_VALIDH
MPH1_CLK/ DR_CLK	I	ULPI—use as MPH1_CLK or DR_CLK Serial—use as MPH1_CLK or DR_CLK UTMI—use as DR_CLK
MPH0_D0_ENABLEN/ DR_D8_CHGVBUS	I/O	ULPI—use as MPH0_D0 Serial—use as MPH0_ENABLEN Serial OTG—use as DR_CHGVBUS UTMI—use as DR_D8

Table 16-2. USB External Signals (continued)

Signal	I/O	Description
MPH0_D1_SER_TXD/ DR_D9_DCHGVBUS	I/O	ULPI—use as MPH0_D1 Serial—use as MPH0_SER_TXD Serial OTG—use as DR_DCHGVBUS UTMI—DR_D9
MPH0_D2_VMO_SE0/ DR_D10_DPPD	I/O	ULPI—use as MPH0_D2 Serial—use as MPH0_VMO_SE0 Serial OTG—use as DR_DPPD UTMI—use as DR_D10
MPH0_D3_SPEED/ DR_D11_DMPD	I/O	ULPI—use as MPH0_D3 Serial—use as MPH0_SPEED Serial OTG—use as DR_DMPD UTMI—use as DR_D11
MPH0_D4_DP/ DR_D12_VBUS_VLD	I/O	ULPI—use as MPH0_D4 Serial—use as MPH0_DP Serial OTG—use as DR_VBUS_VLD UTMI—use as DR_D12
MPH0_D5_DM/ DR_D13_SESS_END	I/O	ULPI—use as MPH0_D5 Serial—use as MPH0_DM Serial OTG—-use as DR_SESS_END UTMI—use as DR_D13
MPH0_D6_SER_RCV/ DR_D14	I/O	ULPI—use as MPH0_D6 Serial—use as MPH0_SER_RCV UTMI—use as DR_D14
MPH0_D7_DRVVBUS/ DR_D15_IDPULLUP	I/O	ULPI—use as MPH0_D7 Serial—use as MPH0_DRVVBUS Serial OTG—use as DR_IDPULLUP UTMI—use as DR_D15
MPH0_NXT/ DR_RX_ACTIVE_ID	I	ULPI—use as MPH0_NXT Serial OTG—use as DR_RX_ACTIVE_ID UTMI—use as DR_RX_ACTIVE_ID
MPH0_DIR_DPPULLUP/ DR_RESET	I/O	ULPI—use as MPH0_DIR Serial—use as MPH0_DPPULLUP UTMI—use as DR_RESET
MPH0_STP_SUSPEND/ DR_TX_READY	I/O	ULPI—use as MPH0_STP Serial—use as MPH0_SUSPEND UTMI—use as DR_TX_READY
MPH0_PWRFAULT/ DR_RX_VALIDH	I	ULPI—use as MPH0_PWRFAULT Serial—use as MPH0_PWRFAULT UTMI—use as DR_RX_VALIDH
MPH0_PCTL0/ DR_LINE_STATE0	I/O	ULPI—use as MPH0_PCTL0 Serial—use as MPH0_PCTL0 UTMI—use as DR_LINE_STATE0
MPH0_PCTL1/ DR_LINE_STATE1	I/O	ULPI—use as MPH0_PCTL1 Serial—use as MPH0_PCTL1 UTMI—use as DR_LINE_STATE1

Table 16-2. USB External Signals (continued)

Signal	I/O	Description
MPH0_CLK/ DR_RX_VALID	I	ULPI—use as MPH0_CLK Serial—use as MPH0_CLK UTMI—use as DR_RX_VALID
DR_TERM_SEL	O	UTMI—use as DR_TERM_SEL
DR_OPMODE0	O	UTMI—use as DR_OPMODE0
DR_OPMODE1	O	UTMI—use as DR_OPMODE1

In the following sections, the signal descriptions are grouped by interface type. When a signal has multiple functions, each function is described in the appropriate interface section, therefore a signal can have multiple entries.

16.2.1 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 bi-directional, generic data bus that carries USB and register data. This interface minimizes pin count requirements for external PHYs. Table 16-3 describes the signals for the ULPI interface.

NOTE

The ULPI signals are multiplexed with the other PHY interfaces. This section describes the signal functions when in ULPI mode; refer to the other PHY sections for descriptions of the functionally in those modes. Note that the prefix xx_ in the ULPI signals naming stands for MPH0_ or MPH1_ or DR_, as there are three ports that can be activated in ULPI mode.

Table 16-3. ULPI Signal Descriptions

Signal	I/O	Description	
xx_DIR	I	Direction. xx_DIR controls the direction of the data bus. When the PHY has data to transfer to USB port, it drives xx_DIR high to take ownership of the bus. When the PHY has no data to transfer it drives xx_DIR low and monitors the bus for link activity. The PHY pulls xx_DIR high whenever the interface cannot accept data from the link. For example, when the internal PHY PLL is not stable	
		State Meaning	Asserted—PHY has data to transfer to the link. Negated—PHY has no data to transfer.
		Timing	Synchronous to xx_CLK.
xx_NXT	I	Next data. The PHY asserts xx_NXT to throttle the data. When USB port is sending data to the PHY, xx_NXT indicates when the current byte has been accepted by the PHY. USB port places the next byte on the data bus in the following clock cycle. When the PHY is sending data to USB port, xx_NXT indicates when a new byte is available for USB port to consume.	
		State Meaning	Asserted—PHY is ready to transfer byte. Negated—PHY is not ready.
		Timing	Synchronous to xx_CLK.

Table 16-3. ULPI Signal Descriptions (continued)

Signal	I/O	Description	
xx_STP	O	Stop. xx_STP indicates the end of a transfer on the bus.	
		State Meaning	Asserted—USB asserts this signal for 1 clock cycle to stop the data stream currently on the bus. If USB port is sending data to the PHY, xx_STP indicates the last byte of data was previously on the bus. If the PHY is sending data to USB port , xx_STP forces the PHY to end its transfer, de-assert xx_DIR and relinquish control of the data bus to the USB port. Negated—Indicates normal operation.
		Timing	Synchronous to xx_CLK.
xx_PWRFAULT	I	Power fault. xx_PWRFAULT indicates whether a power fault occurred on the USB port Vbus.	
		State Meaning	Asserted—Indicates that a Vbus fault occurred. Applications that support power switching must shut down Vbus power. Negated—Indicates normal operation.
		Timing	Synchronous to xx_CLK.
xx_D*n*	I/O	Data bit *n*. xx_D*n* is bit *n* of the 8-bit (xx_D0–xxD8), bi-directional data bus used to carry, USB, register, and interrupt data between the USB port controller and the PHY.	
		State Meaning	Asserted—Data bit *n* is 1. Negated—Data bit *n* is 0.
		Timing	Synchronous to xx_CLK
xx_PCTL*n*	O	Port control *n*. xx_PCTL*n* is bit n of the 2-bit (xx_PCTL0–xx_PCTL1). xx_PCTL*n* controls the port status indicator LED 0 for USB port when in host mode.	
		State Meaning	Asserted—LED on. Negated—LED off.
		Timing	Synchronous to xx_CLK.

16.2.2 UTMI Interface

UTMI was developed to specify a standard interface between USB 2.0 controllers and USB2.0 PHYs. This interface is made available to support applications that may use a UTMI compliant PHY. UTMI provides adequate signalling for device only applications. Note that UTMI is only available to the DR module in device mode. UTMI+ extensions are not supported by the USB controller on this device; however, the functionality added by UTMI+ is available in the ULPI interface. Table 16-4 describes the signals for the UTMI interface.

NOTE

Some of the UTMI signals are muxed with the other PHY interfaces. This section describes the function when in UTMI mode; refer to the other PHY sections for descriptions of the functionality in those modes.

Table 16-4. UTMI Signal Descriptions

Signal	I/O	Description	
DR_RX_ACTIVE_ID	I	Receive active. Input from the PHY indicating that the receive state machine has detected SYNC and is active. DR_RX_ACTIVE_ID is negated after a bit stuff error or an EOP is detected. Note that DR_RX_ACTIVE_ID is multiplexed with the MPH0_NXT signal.	
		State Meaning	Asserted—PHY is receiving data on the USB port. Negated—PHY is not receiving data on the USB port.
		Timing	Synchronous to DR_CLK.
DR_D*n*	I/O	Data bit *n*. DR_D*n* is bit *n* of the 16-bit (DR_D0–DR_D15), bi-directional data bus used to carry, USB, register, and interrupt data between the USB port controller and the PHY.	
		State Meaning	Asserted—Data bit *n* is 1. Negated—Data bit *n* is 0.
		Timing	Synchronous to DR_CLK.
DR_RX_ERROR	I	Receive error. Indicate whether a receive error has been detected by the PHY.	
		State Meaning	Asserted—Error detected. Negated—Normal operation.
		Timing	Synchronous to DR_CLK.
DR_RX_VALID	I	Receive valid. Indicates whether receive data bits 7 to 0 have valid data.	
		State Meaning	Asserted—Receive data is valid. Negated—Receive data is not valid.
		Timing	Synchronous to DR_CLK.
DR_RX_VALIDH	I	Receive valid high. Indicates whether receive data bits 15 to 8 have valid data. This signal is only used in 16-bit mode.	
		State Meaning	Asserted—Receive data is valid. Negated—Receive data is not valid.
		Timing	Synchronous to DR_CLK.
DR_TX_READY	I	Transmit ready. Used to insert wait states during transmit bus transactions. If DR_TX_READY is negated, data must be held on the UTMI data bus. If DR_TX_READY is asserted, the PHY will store the data on the next clock edge. DR_TX_READY is only valid when DR_TX_VALID is asserted.	
		State Meaning	Asserted—OK to transmit data. Negated—Wait state.
		Timing	Synchronous to DR_CLK.
DR_TX_VALID	O	Transmit valid. Indicates whether transmit data bits 7 to 0 have valid data.	
		State Meaning	Asserted—Transmit data is valid. Negated—Transmit data is not valid.
		Timing	Synchronous to DR_CLK.
DR_TX_VALIDH	O	Transmit valid high. Indicates whether transmit data bits 15 to 8 have valid data. This signal is only used in 16-bit mode.	
		State Meaning	Asserted—Transmit data is valid. Negated—Transmit data is not valid.
		Timing	Synchronous to DR_CLK.

Table 16-4. UTMI Signal Descriptions (continued)

Signal	I/O	Description	
DR_LINE_STATE[1:0]	I	USB line status. Used by the controller for detecting reset, speed signaling, packet timing, and to transition from one behavior to another. These input signals reflect the current state of the single-ended receivers. They are combinatorial until a usable PHY clock is available then they are synchronized to the PHY clock. They directly reflect the current state of the USB DP and DM signals.	
		State Meaning	00 Single-ended 0. 01 J state. 10 K state. 11 Single-ended 1.
		Timing	Synchronous to DR_CLK.
DR_OPMODE[1:0]	O	Operational mode. Used to select various operating modes of the PHY. Note that DR_OPMODE[1] is multiplexed with the TSEC2_TXD[5] signal, and DR_OPMODE[0] is multiplexed with the TSEC2_TXD[4] signal.	
		State Meaning	00 Normal operation (The PHY automatically appends the SYNC and EOP pattern). 01 Non-driving. 10 Disable bit stuffing and NRZI encoding. 11 Normal operation without automatic generation of SYNC and EOP. NRZI encoding is always enabled. Bit stuffing depends on the value of TxBitstuffEnable and TxBitstuffEnableH.
		Timing	Synchronous to DR_CLK.
DR_RESET	O	Reset. Used to reset the PHY.	
		State Meaning	Asserted—Reset PHY. Negated—Normal operation.
		Timing	Synchronous to DR_CLK.
DR_TERM_SEL	O	Termination select. Used with DR_XCVR_SEL to control termination related operations**.** Note that DR_TERM_SEL is multiplexed with the TSEC2_TXD6 signal.	
		State Meaning	Asserted—Full-speed termination. Negated—High Speed termination.
		Timing	Synchronous to DR_CLK.
DR_XCVR_SEL	O	Transceiver select. Used by the USB controller to control the signalling speed.	
		State Meaning	Asserted—Full speed (12 Mbps). Negated—High speed (480 Mbps).
		Timing	Synchronous to DR_CLK.
DR_SESS_VLD	I	Session valid. Used to indicate whether the Vbus voltage is at a valid level for operation.	
		State Meaning	Asserted— Vbus voltage is in the valid range as specified in the USB2.0 specification. Negated—Vbus voltage is outside the valid range as specified in the USB2.0 specification.
		Timing	Synchronous to DR_CLK.
$\overline{\text{DR_SUSPEND}}$	O	Suspend (active low). Used to place the FS/LS transceivers in low power mode when a suspend condition is detected.	
		State Meaning	Asserted (low)—Indicates PHY should enter low power mode. Negated (high)—Indicates PHY should be in normal operation.
		Timing	Synchronous to DR_CLK.

16.2.3 Serial Interface

For full-speed/low-speed USB applications the operating frequency is low enough to allow data recovery to be handled digitally and level translation through a simple transceiver (PHY). The serial transceiver interface signals are brought out of the IP block to support just such legacy USB1.1 FS/LS transceivers. Table 16-5 describes the signals for the serial interface. Note that the timing of the serial interface is asynchronous, however, the USB controller must be connected to a 60 MHz input clock in order to generate the serial output data or to over-sample the serial input data. For the DR module, a 60 MHz input clock must be connected to the DR_CLK signal. For the MPH module, if only MPH0 is used, a 60 MHz input clock must be connected to the MPH0_CLK signal. If only MPH1 is used or if both MPH0 and MPH1 are used, a 60 MHz input clock must be connected to both the MPH0_CLK and MPH1_CLK signals.

NOTE

Some of the serial signals are muxed with the other PHY interfaces. This section describes the function when in serial mode; refer to the other PHY sections for descriptions of the functionally in those modes. Note that the prefix xx_ in the serial signals naming stands for MPH0_ or MPH1_ or DR_, as there are three ports that can be activated in serial mode.

Table 16-5. Serial Detailed Signal Descriptions

Signal	I/O	Description	
xx_SER_RCV	I	Receive. USB FS/LS serial differential input signal in single-ended form.	
		State Meaning	Asserted—J state; Data 1 Negated—K state; Data 0
		Timing	Asynchronous.
$\overline{\text{xx_ENABLEN}}$	O	Enable. Active low output enable signal for a FS/LS transceiver (PHY).	
		State Meaning	Asserted (low)—PHY is in transmit mode Negated (high)—PHY is in receive mode
		Timing	Asynchronous.
xx_SER_TXD	O	Transmit data. Supplies the single-ended data value to the PHY. This signal is valid only when single-ended zero (xx_VMO_SE0) is low.	
		State Meaning	Asserted—Data 1 Negated—Data 0
		Timing	Asynchronous.
xx_VMO_SE0	O	Single-ended zero. Indicates to the FS/LS transceiver that is should send a single-ended zero.	
		State Meaning	Asserted—Single-ended zero Negated—Normal data
		Timing	Asynchronous.
xx_DP	I	Data plus. Reflects the state of the USB DP signal.	
		State Meaning	Asserted—Data 1 Negated—Data 0
		Timing	Asynchronous.
xx_DM	I	Data minus. Reflects the state of the USB DM signal.	
		State Meaning	Asserted—Data 1 Negated—Data 0
		Timing	Asynchronous.

Table 16-5. Serial Detailed Signal Descriptions (continued)

Signal	I/O	Description	
xx_SPEED	O	Speed. Used with some FS/LS transceivers to select LS and FS edge rates.	
		State Meaning	Asserted—Full speed Negated—Low speed
		Timing	Asynchronous.
xx_DPPULLUP	O	Data plus pull-up. Controls the pull-up termination on the USB DP signal in FS only applications.	
		State Meaning	Asserted—Pull up enabled Negated—Pull up disabled
		Timing	Asynchronous.
xx_PWRFAULT	I	Power fault. Indicates whether a Vbus fault has occurred. Controllers that support power switching must shut down Vbus power. This signal is only used in host mode.	
		State Meaning	Asserted—Over current condition on Vbus Negated—Normal operation
		Timing	Asynchronous.
xx_PCTL0	O	Port control 0. Controls the port status indicator LED 0 when in host mode.	
		State Meaning	Asserted—LED on Negated—LED off
		Timing	Asynchronous.
xx_PCTL1	O	Port control 1. Controls the port status indicator LED 1 when in host mode.	
		State Meaning	Asserted—LED on Negated—LED off
		Timing	Asynchronous.
DR_SESS_VLD (DR only)	I	Session valid. Indicates whether the Vbus voltage is at a valid level for operation. Note that this signal is only applicable to the DR module.	
		State Meaning	Asserted— Vbus voltage is in the valid range as specified in the USB2.0 specification. Negated—Vbus voltage is outside the valid range as specified in the USB2.0 specification.
		Timing	Asynchronous.
xx_SUSPEND	O	Suspend (active high). Used to place the FS/LS transceivers in low power mode when a suspend condition is detected. This signal is active high in serial mode.	
		State Meaning	Asserted—Indicates PHY should enter low power mode. Negated—Indicates PHY should be in normal operation.
		Timing	Asynchronous.
xx_DRVVBUS	O	Drive Vbus. Used to enable/disable power (VBus) on devices that support port power switching. This signal is not applicable to device-only applications.	
		State Meaning	Asserted—Vbus power enabled. Negated—Vbus power disabled.
		Timing	Asynchronous.

16.2.4 Serial OTG Interface

Table 16-5 describes the signals for the serial OTG interface. Note that only the DR module supports the OTG functionality. Note that the timing of the serial interface is asynchronous, however, the USB controller must be connected to a 60 MHz input clock (use DR_CLK pin) to generate the serial output data or to over-sample the serial input data.

NOTE

Some of the serial signals are muxed with the other PHY interfaces. This section describes the function when in serial mode; refer to the other PHY sections for descriptions of the functionally in those modes.

Table 16-6. Serial OTG Detailed Signal Descriptions

Signal	I/O	Description	
DR_SER_RCV	I	Receive. USB FS/LS serial differential input signal in single-ended form.	
		State Meaning	Asserted—J state; Data 1 Negated—K state; Data 0
		Timing	Asynchronous.
$\overline{\text{DR_ENABLEN}}$	O	Enable. Active low output enable signal for a FS/LS transceiver (PHY).	
		State Meaning	Asserted (low)—PHY is in transmit mode Negated (high)—PHY is in receive mode
		Timing	Asynchronous.
DR_SER_TXD	O	Transmit data. Supplies the single-ended data value to the PHY. This signal is only valid when single-ended zero (DR_VMO_SE0) is low.	
		State Meaning	Asserted—Data 1 Negated—Data 0
		Timing	Asynchronous.
DR_VMO_SE0	O	Single-ended zero. Indicates to the FS/LS transceiver that is should send a single-ended zero.	
		State Meaning	Asserted—Single-ended zero Negated—Normal data
		Timing	Asynchronous.
DR_DP	I	Data plus. Reflects the state of the USB DP signal.	
		State Meaning	Asserted—Data 1 Negated—Data 0
		Timing	Asynchronous.
DR_DM	I	Data minus. Reflects the state of the USB DM signal.	
		State Meaning	Asserted—Data 1 Negated—Data 0
		Timing	Asynchronous.
DR_SPEED	O	Speed. Used with some FS/LS transceivers to select LS and FS edge rates.	
		State Meaning	Asserted—Full speed Negated—Low speed
		Timing	Asynchronous.
DR_DPPULLUP	O	Data plus pull-up. Controls the pull-up termination on the USB DP signal in FS only applications.	
		State Meaning	Asserted—Pullup enabled Negated—Pullup disabled
		Timing	Asynchronous.

Table 16-6. Serial OTG Detailed Signal Descriptions (continued)

Signal	I/O	Description	
DR_PWRFAULT	I	Power fault. Indicates whether a Vbus fault has occurred. Controllers that support power switching must shut down Vbus power. This signal is only used in host mode.	
		State Meaning	Asserted—Over current condition on Vbus Negated—Normal operation
		Timing	Asynchronous.
DR_PCTL0	O	Port control 0. Controls the port status indicator LED 0 when in host mode.	
		State Meaning	Asserted—LED on Negated—LED off
		Timing	Asynchronous.
DR_PCTL1	O	Port control 1. Controls the port status indicator LED 1 when in host mode.	
		State Meaning	Asserted—LED on Negated—LED off
		Timing	Asynchronous.
DR_SESS_VLD	I	Session valid. Indicates whether the peripheral session is valid.	
		State Meaning	Asserted— Vbus > VA_sess_valid threshold as specified in the USB2.0 OTG supplement specification. Negated—Vbus < VA_sess_valid threshold as specified in the USB2.0 OTG supplement specification.
		Timing	Asynchronous.
DR_SUSPEND	O	Suspend (active high). Used to place the FS/LS transceivers in low power mode when a suspend condition is detected. This signal is active high in serial mode.	
		State Meaning	Asserted—Indicates PHY should enter low power mode. Negated—Indicates PHY should be in normal operation.
		Timing	Asynchronous.
DR_DRVVBUS	O	Drive Vbus. Used to enable/disable power (VBus) on devices that support port power switching. This signal is not applicable to device-only applications.	
		State Meaning	Asserted—Vbus power enabled Negated—Vbus power disabled
		Timing	Asynchronous.
DR_IDPULLUP	O	ID pull-up. Used to enable/disable an ID pullup resistor in OTG operation	
		State Meaning	Asserted—Enabled Negated—Disabled
		Timing	Asynchronous.
DR_VBUS_VLD	I	Vbus valid. Indicates whether Vbus voltage is at a valid level for operation.	
		State Meaning	Asserted— Vbus > VA_vbus_valid threshold as specified in the USB2.0 OTG supplement specification. Negated—Vbus < VA_vbus_valid threshold as specified in the USB2.0 OTG supplement specification.
		Timing	Asynchronous.
DR_SESS_END	I	Session end. Indicates whether peripheral session has ended	
		State Meaning	Asserted—Vbus < VA_sess_end threshold as specified in the USB2.0 OTG supplement specification Negated—Vbus > VA_sess_end threshold as specified in the USB2.0 OTG supplement specification.
		Timing	Asynchronous.

Table 16-6. Serial OTG Detailed Signal Descriptions (continued)

Signal	I/O	Description	
DR_RX_ACTIVE_ID	I	ID. Indicates the state of the ID pin on the USB mini receptacle. This signal makes it possible to determine which kind of plug is connected (Mini-A or Mini-B).	
		State Meaning	Asserted—Mini-B connected Negated—Mini-A connected
		Timing	Asynchronous.
DR_CHGVBUS	O	Charge Vbus. If DR_CHGVBUS is asserted Vbus is pulled up through a resistor. This is done to initiate SRP. This signal is needed only for OTG applications.	
		State Meaning	Asserted—Pullup enabled Negated—Pullup disabled
		Timing	Asynchronous.
DR_DCHGVBUS	O	Discharge Vbus. If DR_DCHGVBUS is asserted high Vbus is pulled down through a resistor to ground. This is needed to discharge Vbus before initiating SRP. B-peripherals use this signal to ensure that Vbus is at a low enough voltage before starting SRP. This signal is only needed for OTG applications.	
		State Meaning	Asserted—Pulldown enabled Negated—Pulldown disabled
		Timing	Asynchronous.
DR_DMPD	O	Data minus pull-down. Used to switch on the 15-KΩ pull-down resistors on the USB DM on host applications. For a peripheral, DR_DMPD should be negated (low). For a host controller, DR_DMPD should be asserted (high).	
		State Meaning	Asserted—Pulldown enabled Negated—Pulldown disabled
		Timing	Asynchronous.
DR_DPPD	O	Data plus pull-down. Used to switch on the 15 kOhm pull-down resistors on the USB DP on host applications. For a peripheral, DR_DPPD should been negated (low). For a host controller, DR_DPPD should be asserted (high).	
		State Meaning	Asserted—Pulldown enabled Negated—Pulldown disabled
		Timing	Asynchronous.

16.2.5 PHY Clocks

The xx_CLK inputs (where xx is MPH0, MPH1, or DR) provide the clocking signals for all PHY interfaces, including ULPI, UTMI and serial. The clock is 60 MHz for ULPI, serial and UTMI 8-bit modes. A 30 MHz clock is required for UTMI 16-bit mode. Detailed clock specifications are given in the appropriate hardware specifications document.

16.3 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 16-7.

Table 16-7. USB Interface Memory Map

Offset	Register	Access	Reset	Section/Page
USB MPH Controller Registers				
0x2_2000–0x2_20FF	Reserved, should be cleared	—	—	—
0x2_2100	CAPLENGTH—Capability register length	R	0x40	16.3.1.1/16-19
0x2_2102	HCIVERSION—Host interface version number	R	0x0100	16.3.1.2/16-19
0x2_2104	HCSPARAMS—Host crtl. structural parameters	R	0x0121_0012	16.3.1.3/16-20
0x2_2108	HCCPARAMS—Host crtl. capability parameters	R	0x0000_0006	16.3.1.4/16-21
0x2_2140	USBCMD—USB command	R/W	0x0008_*n*B*n*0	16.3.2.1/16-23
0x2_2144	USBSTS—USB status	Mixed	0x0000_0000	16.3.2.2/16-26
0x2_2148	USBINTR—USB interrupt enable	R/W	0x0000_0000	16.3.2.3/16-28
0x2_214C	FRINDEX—USB frame index	R/W	0x0000_*nnnn*	16.3.2.4/16-30
0x2_2154	PERIODICLISTBASE—Frame list base address	R/W	0x*nnnn*_0000	16.3.2.6/16-31
0x2_2158	ASYNCLISTADDR—Next asynchronous list addr	R/W	0x0000_0000	16.3.2.8/16-32
0x2_215C	ASYNCTTSTS—Asynchronous buffer status for embedded TT	TBD	0x0000_0000	16.3.2.10/16-34
0x2_2160	BURSTSIZE—Programmable burst size	R/W	0x000_1010	16.3.2.11/16-34
0x2_2164	TXFILLTUNING—Host TT transmit pre-buffer packet tuning	R/W	0x0002_0000	16.3.2.12/16-35
0x2_2168	TXTTFILLTUNING—Host TT transmit pre-buffer packet tuning	R/W	0x0000_0000	16.3.2.13/16-37
0x2_2170	ULPI VIEWPORT—ULPI Register Access	R/W	0x0000_0000	16.3.2.14/16-37
0x2_2180	CONFIGFLAG—Configured flag register	R	0x0000_0001	16.3.2.15/16-39
0x2_2184	PORTSC1—Port status/control 1	R/W	0x8C00_0001	16.3.2.16/16-39
0x2_2188	PORTSC2—Port status/control 2	R/W	0x8C00_0001	16.3.2.16/16-39
0x2_21A8	USBMODE—USB device mode	R/W	0x0000_0003	16.3.2.18/16-47
0x2_2400	SNOOP1—Snoop 1	R/W	0x0000_0000	16.3.2.26/16-53
0x2_2404	SNOOP2—Snoop 2	R/W	0x0000_0000	16.3.2.26/16-53
0x2_2408	AGE_CNT_THRESH—Age count threshold	R/W	0x0000_0000	16.3.2.27/16-54
0x2_2410	SI_CTRL—System interface control	R/W	0x0000_0000	16.3.2.28/16-56
0x2_240C	PRI_CTRL—Priority control	R/W	0x0000_0000	16.3.2.29/16-56
0x2_2500	CONTROL—Control	R/W	0x0000_0000	16.3.2.30/16-57

Table 16-7. USB Interface Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x2_2504–0x2_2FFF	Reserved, should be cleared	—	—	—
USB DR Controller Registers				
0x2_3000–0x2_30FF	Reserved, should be cleared	—	—	—
0x2_3100	CAPLENGTH—Capability register length	R	0x40	16.3.1.1/16-19
0x2_3102	HCIVERSION—Host interface version number	R	0x0100	16.3.1.2/16-19
0x2_3104	HCSPARAMS—Host crtl. structural parameters	R	0x0111_0011	16.3.1.3/16-20
0x2_3108	HCCPARAMS—Host crtl. capability parameters	R	0x0000_0006	16.3.1.4/16-21
0x2_3120	DCIVERSION—Device interface version number	R	0x0001	16.3.1.5/16-22
0x2_3124	DCCPARAMS—Device controller parameters	R	0x0000_0186	16.3.1.6/16-22
0x2_3140	USBCMD—USB command	R/W	0x0008_*n*B*n*0	16.3.2.1/16-23
0x2_3144	USBSTS—USB status	R/W	0x0000_0000	16.3.2.2/16-26
0x2_3148	USBINTR—USB interrupt enable	R/W	0x0000_0000	16.3.2.3/16-28
0x2_314C	FRINDEX—USB frame index	R/W	0x0000_*nnnn*	16.3.2.4/16-30
0x2_3154	PERIODICLISTBASE—Frame list base address	R/W	0x*nnnn*_0000	16.3.2.6/16-31
0x2_3154	DEVICEADDR—USB device address	R/W	0x0000_0000	16.3.2.7/16-32
0x2_3158	ASYNCLISTADDR—Next asynchronous list addr (host mode)	R/W	0x0000_0000	16.3.2.8/16-32
0x2_3158	ENDPOINTLISTADDR—Address at endpoint list (device mode)	R/W	0x0000_0000	16.3.2.9/16-33
0x2_3160	BURSTSIZE—Programmable burst size	R/W	0x0000_1010	16.3.2.11/16-34
0x2_3164	TXFILLTUNING—Host TT transmit pre-buffer packet tuning	R/W	0x0002_0000	16.3.2.12/16-35
0x2_3170	ULPI VIEWPORT—ULPI register access	R/W	0x0000_0000	16.3.2.14/16-37
0x2_3180	CONFIGFLAG—Configured flag register	R	0x0000_0001	16.3.2.15/16-39
0x2_3184	PORTSC1—Port status/control	R/W	0x8C00_0000	16.3.2.16/16-39
0x2_31A4	OTGSC—On-The-Go status and control	R/W	0x0000_0001	16.3.2.17/16-44
0x2_31A8	USBMODE—USB device mode	R/W	0x0000_0000	16.3.2.18/16-47
0x2_31AC	ENDPTSETUPSTAT—Endpoint setup status	R/W	0x0000_0000	16.3.2.19/16-48
0x2_31B0	ENDPOINTPRIME—Endpoint initialization	R/W	0x0000_0000	16.3.2.20/16-48
0x2_31B4	ENDPTFLUSH—Endpoint de-initialize	R/W	0x0000_0000	16.3.2.21/16-49
0x2_31B8	ENDPTSTATUS—Endpoint status	R	0x0000_0000	16.3.2.22/16-50
0x2_31BC	ENDPTCOMPLETE—Endpoint complete	R/W	0x0000_0000	16.3.2.23/16-50
0x2_31C0	ENDPTCTRL0—Endpoint control 0	R/W	0x0080_0080	16.3.2.24/16-51

Table 16-7. USB Interface Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x2_31C4	ENDPTCTRL1—Endpoint control 1	R/W	0x0000_0000	16.3.2.25/16-52
0x2_31C8	ENDPTCTRL2—Endpoint control 2	R/W	0x0000_0000	16.3.2.25/16-52
0x2_31CA	ENDPTCTRL3—Endpoint control 3	R/W	0x0000_0000	16.3.2.25/16-52
0x2_31D0	ENDPTCTRL4—Endpoint control 4	R/W	0x0000_0000	16.3.2.25/16-52
0x2_31D4	ENDPTCTRL5—Endpoint control 5	R/W	0x0000_0000	16.3.2.25/16-52
0x2_3400	SNOOP1—Snoop 1	R/W	0x0000_0000	16.3.2.26/16-53
0x2_3404	SNOOP2—Snoop 2	R/W	0x0000_0000	16.3.2.26/16-53
0x2_3408	AGE_CNT_THRESH—Age count threshold	R/W	0x0000_0000	16.3.2.27/16-54
0x2_340C	PRI_CTRL—Priority control	R/W	0x0000_0000	16.3.2.29/16-56
0x2_3410	SI_CTRL—System interface control	R/W	0x0000_0000	16.3.2.28/16-56
0x2_3500	CONTROL—Control	R/W	0x0000_0000	16.3.2.30/16-57
0x2_3504–0x2_3FFF	Reserved, should be cleared	—	—	—

The following sections provide details about the registers in the USB memory map. Because there are many identical registers in the MPH and DR modules, the common register descriptions only note the differences between modules. Where there are registers unique to a module, those registers will be described as MPH-only or DR-only.

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 modules, 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.

16.3.1 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.

16.3.1.1 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 16-14 shows CAPLENGTH.

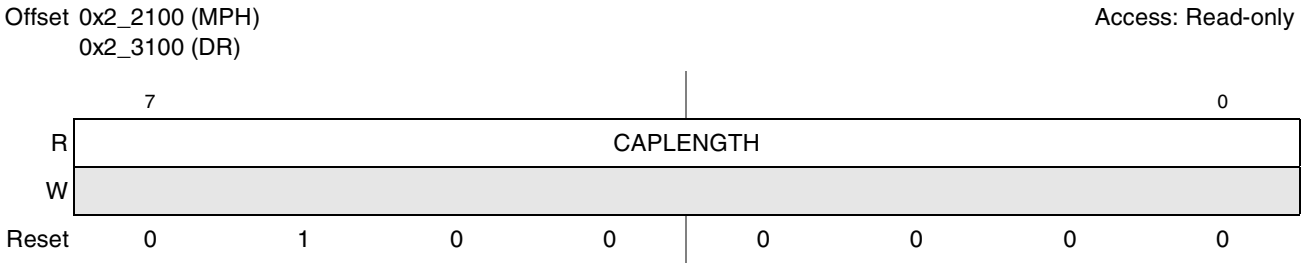


Figure 16-2. Capability Registers Length (CAPLENGTH)

Table 16-8 provides bit descriptions for the CAPLENGTH register.

Table 16-8. CAPLENGTH Register Field Descriptions

Bits	Name	Description
7–0	CAPLENGTH	Capability registers length. Value is 0x40.

16.3.1.2 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 16-16 shows the HCIVERSION register.

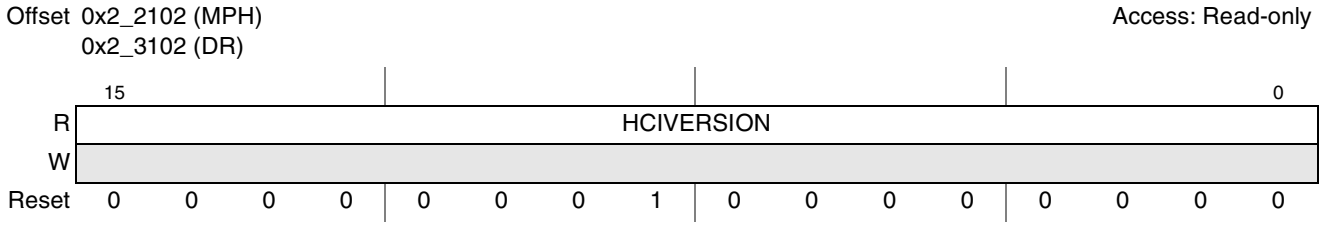


Figure 16-3. Host Controller Interface Version (HCIVERSION)

Table 16-9 provides bit descriptions for the HCIVERSION register.

Table 16-9. HCIVERSION Register Field Descriptions

Bits	Name	Description
15–0	—	EHCI revision number. Value is 0x0100 indicating version 1.0.

16.3.1.3 Host Controller Structural Parameters (HCSPARAMS)

HCSPARAMS contains structural parameters such as the number of downstream ports. Figure 16-18 shows the HCSPARAMS register.

Offset 0x2_2104 (MPH)
0x2_3104 (DR)

Access: Read-only

	31–28	27–24	23–20	19–17	16	15–12	11–8	7–5	4	3–0
R	—	N_TT	N_PTT	—	PI	N_CC	N_PCC	—	PPC	N_PORTS
W										
Reset(MHP)	0 0 0 0	0 0 0 1	0 0 1 0	0 0 0	1	0 0 0 0	0 0 0 0	0 0 0	1	0 0 1 0
Reset(DR)	0 0 0 0	0 0 0 1	0 0 0 1	0 0 0	1	0 0 0 0	0 0 0 0	0 0 0	1	0 0 0 1

Figure 16-4. Host Controller Structural Parameters (HCSPARAMS)

Table 16-10 provides bit descriptions for the HCSPARAMS register.

Table 16-10. HCSPARAMS Register Field Descriptions

Bits	Name	Description
31–28	—	Reserved, should be cleared.
27–24	N_TT	Number of transaction translators. This is a non-EHCI field. This field indicates the number of embedded transaction translators associated the module. This field is always 1. See Section 16.9.1, “Embedded Transaction Translator Function—MPH Module Only.”
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. This bit 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. This field indicates the number of companion controllers associated with the DR controller. This field is always 0.
11–8	N_PCC	Number ports per CC. This field indicates the number of ports supported per internal companion controller. This field is always 0.
7–5	—	Reserved, should be cleared.
4	PPC	Power port control. This bit indicates whether the host controller supports port power control. It is always 1. 1 Ports have power port switches.
3–0	N_PORTS	Number of ports. This field indicates the 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. For the MPH module this is always 0x2; for the DR module it is always 0x1.

16.3.1.4 Host Controller Capability Parameters (HCCPARAMS)

HCCPARAMS identifies multiple mode control (time-base bit functionality) addressing capability. Figure 16-20 shows the HCCPARAMS register.

Offset 0x2_2108 (MPH)
0x2_3108 (DR)

Access: Read-only

	31–16	15–8	7–4	3	2	1	0
R	—	EECP	IST	—	ASP	PFL	ADC
W							
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0	0	1	1	0

Figure 16-5. Host Control Capability Parameters (HCCPARAMS)

Table 16-11 provides bit descriptions for the HCCPARAMS register.

Table 16-11. 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. For the MPH and DR modules, this field is always 0.
7–4	IST	Isochronous scheduling threshold. This field 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.
2	ASP	Asynchronous schedule park capability. This bit indicates if the host controller supports the park feature for high-speed queue heads in the asynchronous schedule. The feature can be disabled or enabled and set to a specific level by using the asynchronous schedule park mode enable and asynchronous schedule park mode count fields in the USBCMD register. For the MPH and DR modules, this field is always 1 (park feature supported).
1	PFL	Programmable frame list flag. Indicates whether system software can specify and use a frame list length less that 1024 elements. Frame list size is configured via the USBCMD register frame list size field. The frame list must always be aligned on a 4K page boundary. This requirement ensures that the frame list is always physically contiguous. This field is always 1.
0	ADC	64-bit addressing capability. This field is always 0; 64-bit addressing is not supported. 0 Data structures use 32-bit address memory pointers

16.3.1.5 Device Controller Interface Version (DCIVERSION)—DR Only, 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 16-22 shows the DCIVERSION register. Note that the MPH module does not have a DCIVERSION register.

Offset 0x2_3120 Access: Read-only

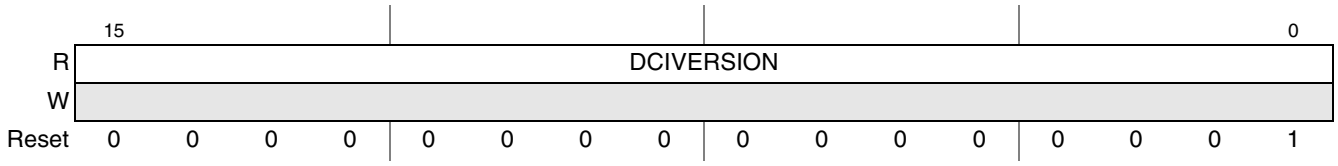


Figure 16-6. Device Interface Version (DCIVERSION)—DR-Only

Table 16-12 provides bit descriptions for the DCIVERSION register.

Table 16-12. DCIVERSION Register Field Descriptions

Bits	Name	Description
15–0	DCIVERSION	Device interface revision number.

16.3.1.6 Device Controller Capability Parameters (DCCPARAMS)—DR Only, Non-EHCI

This register is not defined in the EHCI specification. This register describes the overall host/device capability of the DR module. Figure 16-24 shows the DCCPARAMS register. Note that the MPH module does not have a DCCPARAMS register.

Offset 0x2_3124 Access: Read-only

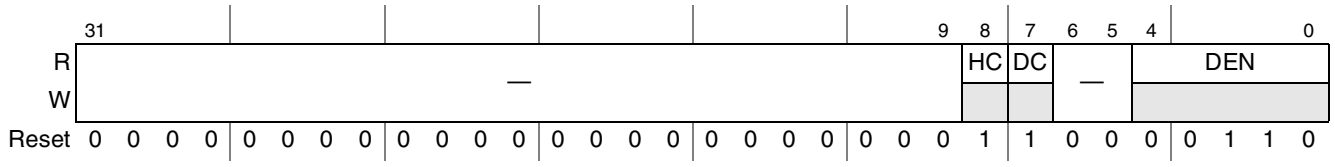


Figure 16-7. Device Control Capability Parameters (DCCPARAMS)—DR-Only

Table 16-13 provides bit descriptions for the DCCPARAMS register.

Table 16-13. DCCPARAMS Register Field Descriptions

Bits	Name	Description
31–9	—	Reserved, should be cleared.
8	HC	Host capable. Always 1, indicating the DR controller can operate as an EHCI compatible USB 2.0 host

Table 16-13. DCCPARAMS Register Field Descriptions (continued)

Bits	Name	Description
7	DC	Device capable. Always 1, indicating the 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 0x6.

16.3.2 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.

16.3.2.1 USB Command Register (USBCMD)

The module executes the command indicated in this register.

Offset 0x2_2140 (MPH)
0x2_3140 (DR)

Access: Mixed

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	—								ITC							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	FS2	ATDTW	SUTW	—	ASPE	—	ASP		LR	IAA	ASE	PSE	FS1	FS0	RST	RS
W																
Reset	0	0	*n*	*n*	1	0	1	1	0	*n*	0	0	0	0	0	0

Figure 16-8. USB Command Register (USBCMD)

Table 16-14. 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 module will issue 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 bit 3–2 below. This is a non-EHCI bit.

Table 16-14. USBCMD Register Field Descriptions (continued)

Bits	Name	Description
14	ATDTW	Add dTD TripWire. This is a non-EHCI bit, that is present on the DR module only. This bit is used as a semaphore when a dTD is added to an active (primed) endpoint. This bit is set and cleared by software. This bit shall also be cleared by hardware when is state machine is hazard region where adding a dTD to a primed endpoint may go unrecognized. More information on the use of this bit is described in Section 16.9.2, “Device Operation—DR Module Only.”
13	SUTW	Setup tripwire. This is a non-EHCI bit, that is present on the DR module only. This bit is used as a semaphore when the 8 bytes of setup data read extracted from a QH by the DCD. If the setup lockout mode is off (See USBMODE) then there exists a hazard when new setup data arrives and the DCD is copying setup from the QH for a previous setup packet. This bit is set and cleared by software and will be cleared by hardware when a hazard exists. More information on the use of this bit is described in Section 16.9.2, “Device Operation—DR Module Only.”
12	—	Reserved, should be cleared.
11	ASPE	Asynchronous schedule park mode enable. This bit defaults to a 1 and is R/W. Software uses this bit to enable or disable park mode. 0 Disabled 1 Enabled
10	—	Reserved, should be cleared.
9–8	ASP	Asynchronous schedule park mode count. This field defaults to 11 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 0x1 to 0x3. Software must not write a zero to this field when Park Mode Enable is a one as this will result in undefined behavior.
7	LR	Light host/device controller reset (OPTIONAL). Not Implemented. Always 0.
6	IAA	Interrupt on async advance doorbell. Used as a doorbell by software to tell the 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 the Interrupt on Async Advance status bit in the USBSTS register. If the Interrupt on sync advance enable bit in the USBINTR register is one, then the host controller will assert an interrupt at the next interrupt threshold. The controller sets this bit to zero after it has set the Interrupt on sync advance status bit in the USBSTS register to one. Software should not write a one to this bit when the asynchronous schedule is inactive. Doing so will yield undefined results. This bit is only used in host mode. Writing a one to this bit when the DR module is in device mode is selected will have undefined results.
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.

Table 16-14. USBCMD Register Field Descriptions (continued)

Bits	Name	Description
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 the frame index register 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 (MPH and DR): When software writes a one to 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 to a one when the HCHalted bit in the USBSTS register is a zero. Attempting to reset an actively running host controller will result in undefined behavior. Device mode (DR only): When software writes a one to this bit, the 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.
0	RS	Run/Stop. Host mode (MPH and DR): When set to a 1, 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 HC Halted bit in the status register indicates when the Host 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, HCHalted in the USBSTS register is a one). Device mode (DR only): Writing a one to this bit will cause the 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 will become disabled upon transitioning into high-speed mode. Software should use this bit to prevent an attach event before the DR controller has been properly initialized. Writing a 0 to this will cause a detach event. 0 Stop 1 Run

16.3.2.2 USB Status Register (USBSTS)

This register indicates various states of each module and any pending interrupts. This register does not indicate status resulting from a transaction on the serial bus. Software clears certain bits in this register by writing a 1 to them (indicated by a w1c in the bit's W cell in the figure).

Offset 0x2_2144 (MPH)
0x2_3144 (DR)

Access: Mixed

	31–24	23–16
R	—	
W		
Reset	All zeros	

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	AS	PS	RCL	HCH	—	ULPII	—	SLI	SRI	URI	AAI	SEI	FRI	PCI	UEI	UI
W								w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	All zeros															

Figure 16-9. USB Status Register (USBSTS)

Table 16-15. 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 controller is not required to immediately disable or enable the asynchronous schedule when software transitions the asynchronous schedule enable bit in the USBCMD register. When this bit and the asynchronous schedule enable bit are 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 controller is not required to immediately disable or enable the periodic schedule when software transitions the periodic schedule enable bit in the USBCMD register. When this bit and the periodic schedule enable bit are 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	HCHalted. This bit is a zero whenever the Run/Stop bit is a one. The controller sets this bit to one after it has stopped executing because of the Run/Stop bit being set to 0, 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. This is a non-EHCI bit. An event completion to the viewport register sets the ULPII bit. If the ULPI enables the ULPIE bit to be set in the USBINTR register, then the USBINT will occur.

Table 16-15. USBSTS Register Field Descriptions (continued)

Bits	Name	Description
9	—	Reserved, should be cleared.
8	SLI	DCSuspend. This is a non-EHCI bit that is present on the DR module only. 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 (MPH and DR): This is a non-EHCI status bit. In host mode, this bit will be set every 125us, 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. Device mode (DR-only): SOF received. When the DR controller detects a Start Of (micro) Frame, this bit will be set. When a SOF is extremely late, the DR controller will automatically set this bit to indicate that an SOF was expected. Therefore, this bit will be set roughly every 1 msec in device FS mode and every 125 msec in HS mode and will be synchronized to the actual SOF that is received. Because the DR controller is initialized to FS before connect, this bit will be set at an interval of 1 msec during the prelude to the connect and chirp. Software writes a 1 to this bit to clear it.
6	URI	USB reset received. This is a non-EHCI bit that is present on the DR module only. When the DR controller detects a USB Reset and enters the default state, this bit will be set. Software can write a 1 to this bit to clear the USB reset received status bit. Only used by the device mode. 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 controller advances the asynchronous schedule by writing a one to the Interrupt on async advance doorbell bit in the USBCMD register. 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 the System Error Enable (SEE) bit in the USBINTR is set, an interrupt will be generated. The interrupt and status bits will remain asserted until cleared by writing a 1 to this bit. Additionally, when in host mode, the RUN/STOP (RS) bit of the USBCMD register is cleared, effectively disabling the controller. For the 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 the Frame List Size field of the USBCMD register) is 1024, the Frame Index Register rolls over every time FRINDEX [1 3] toggles. Similarly, if the size is 512, the controller sets this bit to a one every time FHINDEX [12] toggles. Only used by the host mode.

Table 16-15. USBSTS Register Field Descriptions (continued)

Bits	Name	Description
2	PCI	Host mode (MPH and DR): Port change detect. The controller sets this bit to a one when on any port a connect status occurs, a port enable/disable change occurs, an over current change occurs, or the force port resume bit is set as the result of a J-K transition on the suspended port. Device mode: (DR-only) The DR controller sets this bit to a one 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 the USB reset received bit and the DCSuspend bits respectively. This bit is not EHCI compatible.
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 USBINT bit, if the TD on which the error interrupt occurred also had its interrupt on complete (IOC) bit set. See Section 4.15.1 in EHCI for a complete list of host error interrupt conditions. Also see Table 16-97 in this chapter for more information on device error matrix.For the 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.

16.3.2.3 USB Interrupt Enable Register (USBINTR)

The interrupts to software are enabled with this register. An interrupt is generated when a bit is set and the corresponding interrupt is active. The USB status register (USBSTS) still shows interrupt sources even if they are disabled by the USBINTR register, allowing polling of interrupt events by the software.

Offset 0x2_2148 (MPH)
0x2_3148 (DR)

Access: Read/write

	31															16
R W	—															
Reset	All zeros															

	15			11	10	9	8	7	6	5	4	3	2	1	0
R W	—				ULPIE	—	SLE	SRE	URE	AAE	SEE	FRE	PCE	UEE	UE
Reset	All zeros														

Figure 16-10. USB Interrupt Enable (USBINTR)

Table 16-16. USBINTR Register Field Descriptions

Bits	Name	Description
31–11	—	Reserved, should be cleared.
10	ULPIE	ULPI interrupt enable. This is a non-EHCI bit that is present on both the MPH and DR modules. An event completion to the viewport register sets the ULPII bit. If the ULPI enables the ULPIE bit to be set, then the USBINT will occur. 0 Disable 1 Enable
9	—	Reserved, should be cleared.
8	SLE	Sleep enable. This is a non-EHCI bit that is present on the DR module only. When this bit is a one, and the DCSuspend bit in the USBSTS register transitions, the DR controller will issue an interrupt. The interrupt is acknowledged by software writing a one to the DCSuspend bit. Only used in device mode. 0 Disable 1 Enable
7	SRE	SOF received enable. This is a non-EHCI bit that is present on both the MPH and DR modules. When this bit is a one, and the SOF Received bit in the USBSTS register is a one, the controller will issue an interrupt. The interrupt is acknowledged by software clearing the SOF Received bit. 0 Disable 1 Enable
6	URE	USB reset enable. This is a non-EHCI bit that is present on the DR module only. When this bit is a one, and the USB reset received bit in the USBSTS register is a one, the device controller will issue an interrupt. The interrupt is acknowledged by software clearing the USB Reset Received bit. Only used in device mode. 0 Disable 1 Enable
5	AAE	Interrupt on async advance enable. When this bit is a one, and the Interrupt on Async Advance bit in the USBSTS register is a one, the controller will issue an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing the Interrupt on Async Advance bit. Only used in host mode. 0 Disable 1 Enable
4	SEE	System error enable. When this bit is a one, and the system error bit in the USBSTS register is a one, the controller will issue an interrupt. The interrupt is acknowledged by software clearing the system error bit. 0 Disable 1 Enable
3	FRE	Frame list rollover enable. When this bit is a one, and the frame list rollover bit in the USBSTS register is a one, the controller will issue an interrupt. The interrupt is acknowledged by software clearing the frame list rollover bit. Only used by the host mode. 0 Disable 1 Enable
2	PCE	Port change detect enable. When this bit is a one, and the Port Change Detect bit in the USBSTS register is a one, the controller will issue an interrupt. The interrupt is acknowledged by software clearing the port change detect bit. 0 Disable 1 Enable

Table 16-16. USBINTR Register Field Descriptions (continued)

Bits	Name	Description
1	UEE	USB error interrupt enable. When this bit is a one, and the USBERRINT bit in the USBSTS register is a one, the controller will issue an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing the USBERRINT bit in the USBSTS register. 0 Disable 1 Enable
0	UE	USB interrupt enable. When this bit is a one, and the USBINT bit in the USBSTS register is a one, the DR controller will issue an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing the USBINT bit. 0 Disable 1 Enable

16.3.2.4 Frame Index Register (FRINDEX)

In host mode, this register is used by the controller to index the periodic frame list. The register updates every 125 microseconds (once each microframe). Bits [N–3] are used to select a particular entry in the Periodic Frame List during periodic schedule execution. The number of bits used for the index depends on the size of the frame list as set by system software in the Frame List Size field in the USBCMD register.

This register must be written as a DWord. Byte writes produce-undefined results. This register cannot be written unless the DR controller is in the Halted state as indicated by the HCHalted bit. A write to this register while the Run/Stop hit is set produces undefined results. Writes to this register also affect the SOF value.

In device mode (DR-only), this register is read-only and, the 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 µsec microframe.)

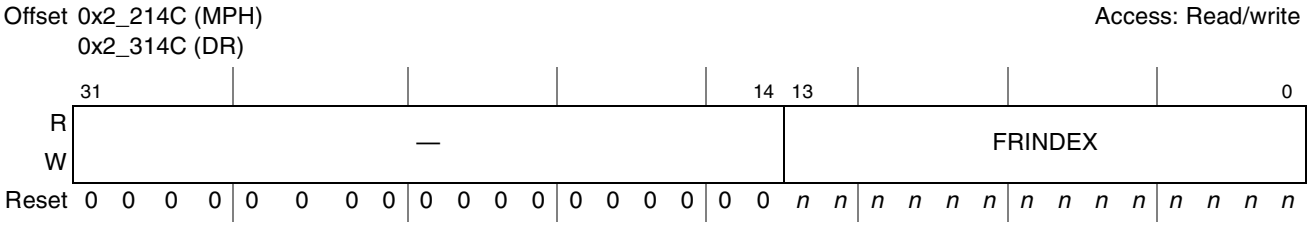


Figure 16-11. USB Frame Index (FRINDEX)

Table 16-17. 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. For the DR module 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 16-18 illustrates values of N based on the value of the Frame List Size in the USBCMD register, when used in host mode.

Table 16-18. FRINDEX N Values

USBCMD[FS]	Frame List Size	FRINDEX N value
000	1024 elements (4096 bytes)	12
001	512 elements (2048 bytes)	11
010	256 elements (1024 bytes)	10
011	128 elements (512 bytes)	9
100	64 elements (256 bytes)	8
101	32 elements (128 bytes)	7
110	16 elements (64 bytes)	6
111	8 elements (32 bytes)	5

16.3.2.5 Control Data Structure Segment Register (CTRLDSSEGMENT)

The CTRLDSSEGMENT register is not implemented on this device.

16.3.2.6 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 on the DR module, 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 16.3.2.7, “Device Address Register (DEVICEADDR)—DR-Only, Non-EHCI,” for more information.

Offset 0x2_2154 (MPH)
0x2_3154 (DR)

Access: Read/write

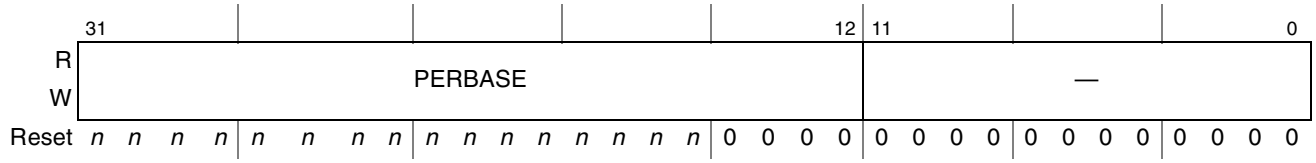


Figure 16-12. Periodic Frame List Base Address (PERIODICLISTBASE)

Table 16-19. 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.

16.3.2.7 Device Address Register (DEVICEADDR)—DR-Only, Non-EHCI

This register is not defined in the EHCI specification. For the DR module 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 on the DR module, 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 16.3.2.6, “Periodic Frame List Base Address Register (PERIODICLISTBASE),” for more information.

Offset 0x2_3154 (DR device mode only)

Access: Read/write

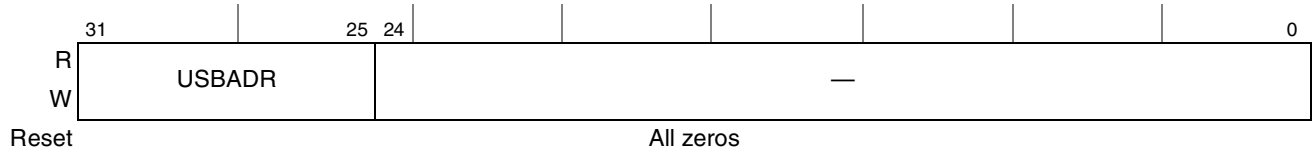


Figure 16-13. Device Address (DEVICEADDR)—DR-Only

Table 16-20. 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.

16.3.2.8 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 on the DR module, 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 16.3.2.9, “Endpoint List Address Register (ENDPOINTLISTADDR)—DR-Only, Non-EHCI,” for more information.

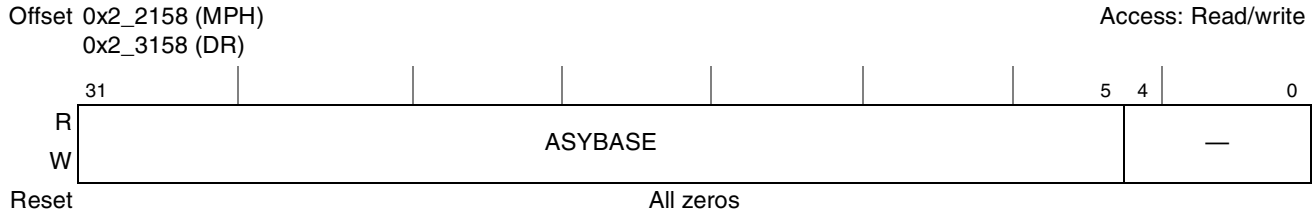


Figure 16-14. Current Asynchronous List Address (ASYNCLISTADDR)

Table 16-21. ASYNCLISTADDR Register Field Descriptions

Bits	Name	Description
31–5	ASYBASE	Link Pointer Low (LPL). These bits correspond to memory address signal [31:5]. This field may only reference a Queue Head (QH). Only used by the host controller.
4–0	—	Reserved, should be cleared.

16.3.2.9 Endpoint List Address Register (ENDPOINTLISTADDR)—DR-Only, Non-EHCI

This register is not defined in the EHCI specification. For the DR module 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 on the DR module, 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 16.3.2.8, “Current Asynchronous List Address Register (ASYNCLISTADDR),” for more information.

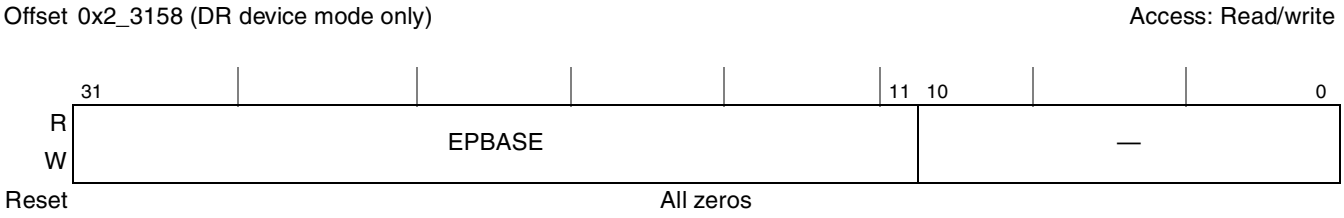


Figure 16-15. Endpoint List Address (ENDPOINTLISTADDR)—DR-Only

Table 16-22. 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.

16.3.2.10 Transaction Translator Asynchronous Buffer Status (ASYNCTTSTS)—MPH-Only, Non-EHCI

This register is not defined in the EHCI specification. This register is used to control the Asynchronous Buffer in the Embedded TT in the MPH module. Because the Unlink Periodic Buffer in the Embedded TT has automatic time-out and clear, the asynchronous buffer can become hung when the host controller driver does not let a complete-split run until completion. This register provides a function similar to that provided by the Clear_TT_Buffer and Get_TT_State.

Offset 0x2_215C Access: Mixed

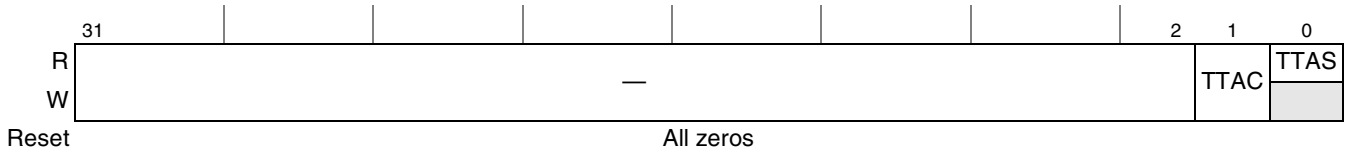


Figure 16-16. TT Asynchronous Buffer Status (ASYNCTTSTS)—MPH-Only

Table 16-23. ASYNCTTSTS Register Field Descriptions

Bits	Name	Description
31–2	—	Reserved, should be cleared.
1	TTAC	Embedded TT async Buffers Clear. This field will clear all pending transactions in the embedded TT Async Buffer(s). The clear will take as much time as necessary to clear buffer without interfering with a transaction in progress. TTAC will return to zero after being set by software only after the actual clear occurs.
0	TTAS	Embedded TT async Buffers Status. 1 One or more transactions are being held in the embedded TT Async Buffers. 0 All outstanding transactions in the embedded TT have been flushed.

16.3.2.11 Master Interface Data Burst Size Register (BURSTSIZE)—Non-EHCI

This register is not defined in the EHCI specification. This register is used to control and dynamically change the burst size used during data movement on the initiator (master) interface.

Offset 0x2_3160 Access: Read/write

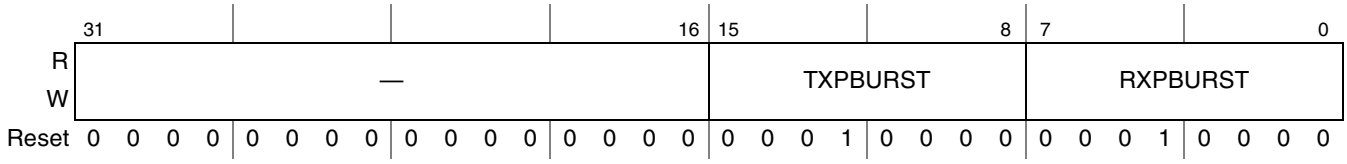


Figure 16-17. Master Interface Data Burst Size (BURSTSIZE)

Table 16-24. BURSTSIZE Register Field Descriptions

Bits	Name	Description
31–16	—	Reserved, should be cleared.
15–8	TXPBURST	Programable TX Burst Length. This register represents the maximum length of a burst in 32-bit words while moving data from system memory to the USB bus. Must not be set to greater that 16. If the Repeat function of the 8349 system bus is activated (see TBD for more information.), then host software should leave this field at the default of 16 words. If the Repeat function is not activated, then host software should set this field to 8. The values 16 and 8, respectively, are recommended. Lower values may be used, but will result in poor utilization of the system bus.
7–0	RXPBURST	Programable RX Burst Length. This register represents the maximum length of a burst in 32-bit words while moving data from the USB bus to system memory. Must not be set to greater than 16. If the Repeat function of the 8349 system bus is activated (see TBD for more information.), then host software should leave this field at the default of 16 words. If the Repeat function is not activated, then host software should set this field to 8. The values 16 and 8, respectively, are recommended. Lower values may be used, but will result in poor utilization of the system bus.

16.3.2.12 Transmit FIFO Tuning Controls Register (TXFILLTUNING)—Non-EHCI

This register is not defined in the EHCI specification. This register is used to control and dynamically change the burst size used during data movement on device DMA transfers. It is only used in host mode.

The fields in this register control performance tuning associated with how the 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 anytime during the pre-fill operation the time remaining the [micro]frame is < T_s then the packet attempt ceases and the packet is tried at a later time. Although this is not an error condition and the module eventually recovers, a mark is made in the scheduler health counter to note the occurrence of a back-off event. When a back-off event is detected, the partial packet fetched may need to be discarded from the latency buffer to make room for periodic traffic that will begin after the next SOF. Too many back-off events can waste bandwidth and power on the system bus and thus should be minimized (not necessarily eliminated). Back-offs can be minimized with use of the TSCHHEALTH (T_{ff}) parameter described below.

Offset 0x2_2164 (MPH)
0x2_3164 (DR)

Access: Read/write

	31–22	21–16	15–13	12–8	7–0
R W	—	TXFIFOTHRES	—	TXSCHHEALTH	TXSCHOH
Reset	0 0 0 0 0 0 0 0 0 0	0 0 0 0 1 0	0 0 0	0 0 0 0 0	0 0 0 0 0 0 0 0

Figure 16-18. Transmit FIFO Tuning Controls (TXFILLTUNING)

Table 16-25. TXFILLTUNING Register Field Descriptions

Bits	Name	Description
31–22	—	Reserved, should be cleared.
21–16	TXFIFOTHRES	FIFO Burst Threshold. These bits 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. The maximum value is 8 if TXPBURST is set to 16, or 16 if TXPBURST is set to 8. For most applications, TXFIFOTHRES can be set to half of the maximum value, or lower. If the value is set too low, underruns may occur periodically. USB is designed to retransmit in the event of underrun, but excessive underrun may result in poor utilization of the link. In this case, TXFIFOTHRES can be raised at the expense of added delay between host OUT transactions (and therefore slightly less link utilization). See the Data Buffer Error count in the appropriate Transfer Descriptor for details on how to determine when underrun is occurring. This value is ignored if the Stream Disable bit in USBMODE register is set. When the SDIS bit 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. These bits 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 1.267µs when a device is connected in High-Speed Mode. The time unit represented in this register is 6.333µs 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 * (BURSTSIZE * 4 bytes-per-word) / (40 * *TimeUnit*), always rounded to the next higher integer. *TimeUnit* is either 1.267 or 6.333 as noted earlier in this description. For example, if TXFIFOTHRES is 5 and BURSTSIZE is 8, then set TXSCHOH to 5*(8*4)/(40*1.267) = 4 for a high-speed link. If this value of TXSCHOH results in a TXSCHHEALTH count of 0 per second, try lowering the value by 1 if optimizing performance is desired. If TXSCHHEALTH exceeds 10 per second, try raising the value by 1. If streaming mode is disabled via the USBMODE register, treat TXFIFOTHRES as the maximum value for purposes of the TXSCHOH calculation.

16.3.2.13 Transaction Translator Transmit FIFO Tuning Controls Register (TXTTFILLTUNING)—MPH-Only, Non-EHCI

This register is not defined in the EHCI specification. This register provides a similar function TXFILLTUNING except there is no equivalent to TXFIFOTHRES because the TT TX latency FIFO is always loaded in a single burst. Even though the pre-fill is a single burst, the back-off event can similarly be minimized using the same methods described in Section 16.3.2.12, "Transmit FIFO Tuning Controls Register (TXFILLTUNING)—Non-EHCI."

Offset 0x2_2168 Access: Read/write

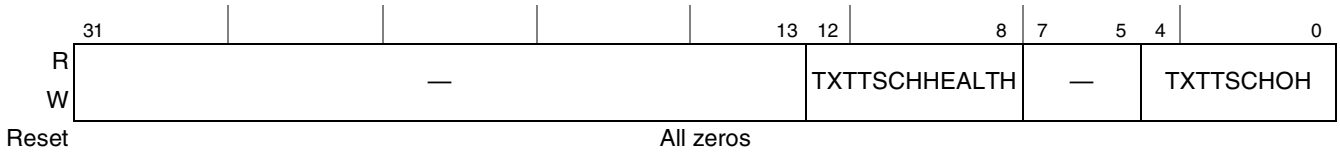


Figure 16-19. TT Transmit FIFO Tuning Controls (TXTTFILLTUNING)—MPH-Only

Table 16-26. TXTTFILLTUNING Register Field Description

Bits	Name	Description
31–13	—	Reserved, should be cleared.
12–8	TXTTSCHHEALTH	TT Scheduler Health Counter. Same description as TXFILLTUNING[TXSCHHEALTH].
7–5	—	Reserved, should be cleared.
4–0	TXTTSCHOH	TT Scheduler Overhead. Same description as TXFILLTUNING[TXSCHOH]. The time unit represented in this register is 6.33 µs. **Note:** TXTTSCHOH=0 or 1 will be sufficient in most systems. The same formula for TXFILLTUNING[TXSCHOH] can be used for TXTTSCHOH. Use a TXFILLTUNING[TXFIFOTHRES] value of 1 for this calculation, and use 6.333 for TimeUnit.

16.3.2.14 ULPI Register Access (ULPI VIEWPORT)

The register provides indirect access to the ULPI PHY register set. Although the controller modules perform access to the ULPI PHY register set, there may be extraordinary circumstances where software may need direct access. Be advised that writes to the ULPI through the ULPI viewport can substantially harm standard USB operations. Currently no usage model has been defined where software should need to execute writes directly to the ULPI. Note that executing read operations though the ULPI viewport should have no harmful side effects to standard USB operations. Also note that if the ULPI interface is not enabled, this register will always read zeros.

ULPI VIEWPORT is shown in Figure 16-31.

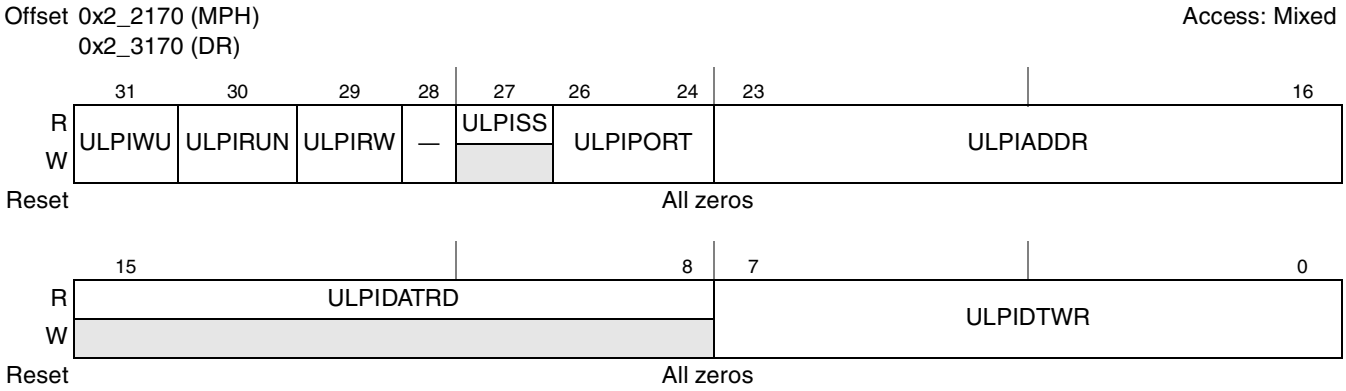


Figure 16-20. ULPI Register Access (ULPI VIEWPORT)

Table 16-27. 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. **Note:** The driver must never execute a wakeup and a read/write operation at the same time.
30	ULPIRUN	ULPI Run. Writing 1 to this bit begins a read/write operation. This bit automatically transitions to 0 after the read/write is complete. Once this bit is set, it can not be cleared by software. **Note:** The driver must never execute a wakeup and a read/write operation at the same time.
29	ULPIRW	This bit selects between running a read or write operation to the ULPI. 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.
23–16	ULPIADDR	When a read or write operation is commanded, the address of the operation is written to this field.
15–8	ULPIDATRD	After a read operation completes, the result is placed in this field.
7–0	ULPIDTWR	When a write operation is commanded, the data to be sent is written to this field.

There are two operations that can be performed with the ULPI viewport, wakeup and read /write operations. The wakeup operation is used to put the ULPI interface into normal operation mode and re-enable the clock if necessary. A wakeup operation is required before accessing the registers when the ULPI interface is operating in low power mode, serial mode, or carkit mode. The ULPI state can be determined by reading the sync state bit (ULPISS). If this bit is set, then the ULPI interface is running in normal operation mode and can accept read/write operations. If the ULPISS is cleared, then read/write

operations will not be able execute. Undefined behavior results if a read or write operation is performed when ULPISS is cleared. To execute a wakeup operation, write all 32-bits of the ULPI Viewport where ULPIPORT is constructed appropriately and the ULPIWU bit is set and the ULPIRUN bit is cleared. Poll the ULPI Viewport until ULPIWU is cleared for the operation to complete.

To execute a read or write operation, write all 32-bits of the ULPI Viewport where ULPIDATWR, ULPIADDR, ULPIPORT, ULPIRW are constructed appropriately and the ULPIRUN bit is set. Poll the ULPI Viewport until ULPIRUN is cleared for the operation to complete. For read operations, ULPIDATRD is valid once ULPIRUN is cleared.

The polling method above can be replaced with interrupts using the ULPI interrupt defined in the USBSTS and USBINTR registers. When a wakeup or read/write operation completes, the ULPI interrupt is set.

16.3.2.15 Configure Flag Register (CONFIGFLAG)

This EHCI register is not used in this implementation. A read from this register returns a constant of a 0x0000_0001 to indicate that all port routings default to this host controller.

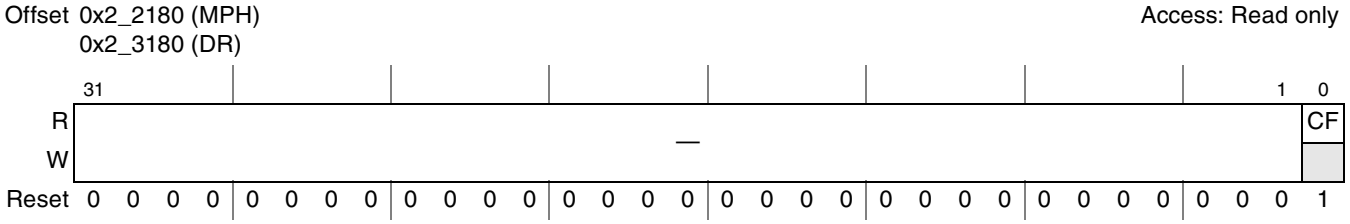


Figure 16-21. Configure Flag Register (CONFIGFLAG)

Table 16-28. CONFIGFLAG Register Field Descriptions

Bits	Name	Description
31–1	—	Reserved.
0	CF	Configure flag. Always 1 indicating all port routings default to this host.

16.3.2.16 Port Status and Control Registers (MPH—PORTSC*n*, DR—PORTSC1)

The MPH module has two PORTSC registers and the DR module has one. These registers are 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.

For the DR module in device mode, the 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 0x2_2184 (MPH PORTSC1), 0x2_2188 (MPH PORTSC2) Access: Mixed
0x2_3184 (DR PORTSC1)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PTS		—	PTW	PSPD		—	PFSC	PHCD	WKOC	WKDS	WLCN	PTC			
W																
Reset (MPH)	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
Reset (DR)	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PIC		PO	PP	LS		—	PR	SUSP	FPR	OCC	OCA	PEC	PE	CSC	CCS
W											w1c		w1c		w1c	
Reset (MPH)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Reset (DR)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-22. Port Status and Control (PORTSC*n*)

Table 16-29. PORTSC*n* 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 UTMI parallel interface 01 Reserved 10 ULPI parallel interface 11 FS/LS serial interface For the MPH ports, UTMI is not supported. This bit is not defined in the EHCI specification.
29	—	Reserved, should be cleared
28	PTW	Parallel Transceiver Width. This register bit is used to control the data bus width of the parallel transceiver interface. This bit defaults to 1 after reset 0 8-bit interface—[60 MHz] UTMI interface. 1 16-bit interface—[30 MHz] UTMI interface. PTW is only valid for UTMI mode (PTS = 00). This bit is not defined in the EHCI specification.
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. This bit is 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.

Table 16-29. PORTSC*n* Register Field Descriptions (continued)

Bits	Name	Description
23	PHCD	PHY Low Power Suspend. This bit is not defined in the EHCI specification. In host mode, 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. For the DR module in device mode, the PHY can be put into Low Power Suspend – when the device is not running (USBCMD Run/Stop=0b) or suspend signaling is detected on the USB. Low power suspend will be cleared automatically when the resume signaling has been detected or when forcing port resume. 0 Normal PHY operation. 1 Signal the PHY to enter low power suspend mode Reading this bit indicates the status of the PHY. **Note:** If there is no clock connected to the xx_CLK signals, then PHCD must be set.
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 (DR-only). 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 (DR-only). 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 Refer to Chapter 7 of the USB Specification Revision 2.0 [3] for details on each test mode.
15–14	PIC	Port Indicator Control. These bits 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. This bit 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.

Table 16-29. PORTSC*n* Register Field Descriptions (continued)

Bits	Name	Description
12	PP	Port Power. This bit 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). For the DR module in a device-only implementation port power control is not necessary, thus PPC and PP = 0.
11–10	LS	Line Status. These bits 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. 01 J-state. 10 K-state. 11 Undefined.
9	—	Reserved, should be cleared
8	PR	Port Reset. In host mode, 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. For the DR module in device mode, this bit is a read only status bit. Device reset from the USB bus is also indicated in the USBSTS register. 1 Port is in Reset. 0 Port is not in Reset. This field is zero if Port Power(PP) is zero.
7	SUSP	Suspend In host mode: The Port Enabled bit (PE) and Suspend (SUSP) bit define the port states as follows: 0x Disable. 10 Enable. 11 Suspend. 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. The module unconditionally sets this bit to zero when software sets the Force Port Resume bit to zero. 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. This field is zero if Port Power(PP) is zero in host mode. For the DR module in device mode: 1 Port in suspend state. 0 Port not in suspend state. Default. In device mode this bit is a read only status bit.

Table 16-29. PORTSC*n* Register Field Descriptions (continued)

Bits	Name	Description
6	FPR	Force Port Resume. This bit is not-EHCI compatible. 1 Resume detected/driven on port. 0 No resume (K-state) detected/driven on port. In host mode: 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, the Port Change Detect bit in the USBSTS register 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. 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. This field is zero if Port Power(PP) is zero in host mode. In Device mode (DR-only): 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 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 will be cleared when the device returns to normal operation. Also, when this bit transitions to a one because a J-to-K transition detected, the Port Change Detect bit in the USBSTS register is also set.
5	OCC	Over-current Change. This bit gets set when there is a change to Over-current Active. Software clears this bit by writing a one to this bit position. For host/OTG implementations the user can provide over-current detection to the USB*n*_PWRFAULT signal for this condition. For device-only implementations (DR-only) this bit must always be 0. 1 Over current detect. 0 No over current.
4	OCA	Over-current Active. This bit will automatically transition from one to zero when the over current condition is removed. For host/OTG implementations the user can provide over-current detection to the USB*n*_PWRFAULT signal for this condition. For device-only implementations (DR-only) this bit must always be 0. 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 (DR-only), the device port is always enabled. (This bit will be zero). 1 Port disabled. 0 No change. This field is zero if Port Power(PP) is zero.
2	PE	Port Enabled/Disabled. In host mode 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. When the port is disabled, (0) downstream propagation of data is blocked except for reset. This field is zero if Port Power(PP) is zero in host mode. In Device Mode (DR-only), the device port is always enabled. (This bit will be one).

Table 16-29. PORTSC*n* Register Field Descriptions (continued)

Bits	Name	Description
1	CSC	Connect Change Status. In host mode, this bit indicates a change has occurred in the port's Current Connect Status. the controller sets this bit for all changes to the port device connect status, even if system software has not cleared an existing connect status change. For example, the insertion status changes twice before system software has cleared the changed condition, hub hardware will be 'setting' an already-set bit (i.e., the bit will remain set). Software clears this bit by writing a one to it. 1 Connect Status has changed. 0 No change. This field is zero if Port Power(PP) is zero. In device mode (DR-only), this bit is undefined.
0	CCS	Current Connect Status. In 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 (DR-only): 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 the Run bit in the USBCMD register. It does not state the device being disconnected or suspended.

16.3.2.17 On-The-Go Status and Control (OTGSC)—DR-Only, Non-EHCI

This register is not defined in the EHCI specification. The 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 and OTG interrupt.

Offset 0x2_31A4 Access: Mixed

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	—	DPIE	1msE	BSEIE	BSVIE	ASVIE	AVVIE	IDIE	—	DPIS	1msS	BSEIS	BSVIS	ASVIS	AVVIS	IDIS
W										w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	—	DPS	1msT	BSE	BSV	ASV	AVV	ID	—			DP	OT	—	VC	VD
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 16-23. OTG Status Control (OTGSC)—DR-Only

Table 16-30. 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.
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. This bit is set when data bus pulsing occurs on DP or DM. Data bus pulsing is only detected when USBMODE.CM = Host (11) and PORTSC(0).PortPower = Off (0). Software must write a one to clear this bit.
21	1msS	1 millisecond timer Interrupt Status. This bit is set once every millisecond. Software must write a one to clear this bit.
20	BSEIS	B Session End Interrupt Status. This bit is set when VBus has fallen below the B session end threshold. Software must write a one to clear this bit.

Table 16-30. OTGSC Register Field Descriptions (continued)

Bits	Name	Description
19	BSVIS	B Session Valid Interrupt Status. This bit is 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. This bit is 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. This bit is 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. This bit is 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.
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.

16.3.2.18 USB Mode Register (USBMODE)—Non-EHCI

This register is not defined in the EHCI specification. This register controls the operating mode of the module.

Offset 0x2_21A8 (MPH)
0x2_31A8 (DR)

Access: Read/Write

	31–5	4	3	2	1	0
R W	—	SDIS	SLOM	—	CM	
Reset(MPH)	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	1	1
Reset (DR)	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0

Figure 16-24. USB Mode (USBMODE)

Table 16-31. USBMODE Register Field Descriptions

Bits	Name	Description
31–5	—	Reserved, should be cleared.
4	SDIS	Stream Disable. In host mode, setting this bit to a 1 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. 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. 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. 1 Active. 0 Inactive. In device mode (DR-only), setting this bit to a 1 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. Note that in High Speed Mode, all packets received will be responded to with a NYET handshake when stream disable is active.
3	SLOM	Setup Lockout Mode (DR-only). For the DR module in device mode, this bit controls behavior of the setup lock mechanism. See Section 16.8.3.5, “Control Endpoint Operation Model.” 1 Setup Lockouts Off(DCD requires use of Setup Data Buffer Tripwire in USBCMD). 0 Setup Lockouts On.
2	—	Reserved, should be cleared.
1–0	CM	Controller Mode. This register can only be written once after reset. If it is necessary to switch modes, software must reset the controller by writing to the RESET bit in the USBCMD register before reprogramming this register. 00 Idle (Default for combination host/device). 01 Reserved 10 Device Controller (Default for device only controller). 11 Host Controller (Default for host only controller). The MPH module defaults to the host controller state; the DR module defaults to the idle state and needs to be initialized to the desired operating mode after reset.

16.3.2.19 Endpoint Setup Status Register (ENDPTSETUPSTAT)—DR-Only, Non-EHCI

This register is not defined in the EHCI specification. This register contains the endpoint setup status. It is only used by the DR module in device mode.

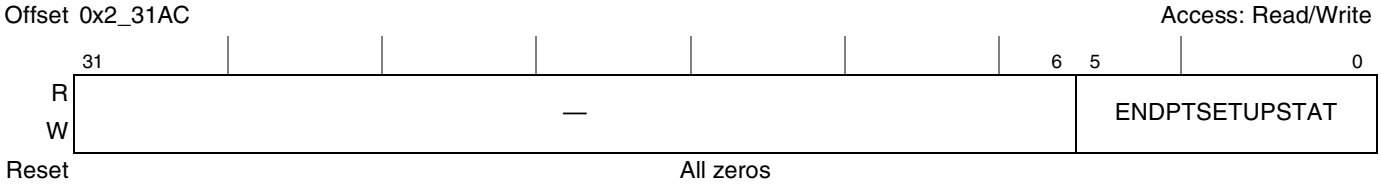


Figure 16-25. Endpoint Setup Status (ENDPTSETUPSTAT)—DR-Only

Table 16-32. ENDPTSETUPSTAT Register Field Descriptions

Bits	Name	Description
31–6	—	Reserved, should be cleared.
5–0	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.

16.3.2.20 Endpoint Initialization Register (ENDPTPRIME)—DR-Only, Non-EHCI

This register is not defined in the EHCI specification. This register is used to initialize endpoints. It is only used by the DR module in device mode.

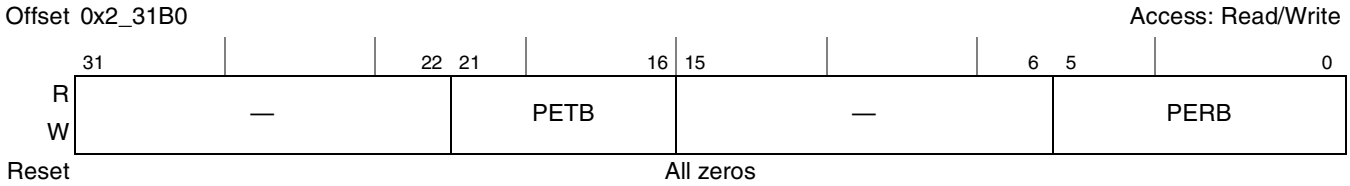


Figure 16-26. Endpoint Initialization (ENDPTPRIME)—DR-Only

Table 16-33. ENDPTPRIME Register Field Descriptions

Bits	Name	Description
31–22	—	Reserved, should be cleared.
21–16	PETB	Prime endpoint transmit buffer. For each endpoint a corresponding bit is used to request that a buffer prepared for a transmit operation in order to respond to a USB IN/INTERRUPT transaction. Software should write a one to the corresponding bit when posting a new transfer descriptor to an endpoint. Hardware will automatically use this bit to begin parsing for a new transfer descriptor from the queue head and prepare a transmit buffer. Hardware will clear this bit when the associated endpoint(s) is (are) successfully primed. PETB[5] (bit 21 of the register) corresponds to endpoint 5. Note that these bits will be momentarily set by hardware during hardware re-priming operations when a dTD is retired, and the dQH is updated.

Table 16-33. ENDPTPRIME Register Field Descriptions (continued)

Bits	Name	Description
15–6	—	Reserved, should be cleared.
5–0	PERB	Prime endpoint receive buffer. For each endpoint, a corresponding bit is used to request a buffer prepare for a receive operation in order to respond to a USB OUT transaction. Software should write a one to the corresponding bit whenever posting a new transfer descriptor to an endpoint. Hardware will automatically use this bit to begin parsing for a new transfer descriptor from the queue head and prepare a receive buffer. Hardware will clear this bit when the associated endpoint(s) is (are) successfully primed. PERB[5] corresponds to endpoint 5. Note that these bits will be momentarily set by hardware during hardware re-priming operations when a dTD is retired, and the dQH is updated.

16.3.2.21 Endpoint Flush Register (ENDPTFLUSH)—DR-Only, Non-EHCI

This register is not defined in the EHCI specification. This register is only used by the DR module in device mode.

Offset 0x2_31B4 Access: Read/Write

	31	22	21	16	15	6	5	0
R W	—		FETB		—		FERB	
Reset	All zeros							

Figure 16-27. Endpoint Flush (ENDPTFLUSH)—DR-Only

Table 16-34. ENDPTFLUSH Register Field Descriptions

Bits	Name	Description
31–22	—	Reserved, should be cleared.
21–16	FETB	Flush endpoint transmit buffer. Writing a one to a bit(s) in this register will cause the associated endpoint(s) to clear any primed buffers. If a packet is in progress for one of the associated endpoints, then that transfer will continue until completion. Hardware will clear this register after the endpoint flush operation is successful. FETB[5] (bit 21 of the register) corresponds to endpoint 5.
15–6	—	Reserved, should be cleared.
5–0	FERB	Flush endpoint receive buffer. Writing a one to a bit(s) will cause the associated endpoint(s) to clear any primed buffers. If a packet is in progress for one of the associated endpoints, then that transfer will continue until completion. Hardware will clear this register after the endpoint flush operation is successful. FERB[5] corresponds to endpoint 5.

16.3.2.22 Endpoint Status Register (ENDPTSTATUS)—DR-Only, Non-EHCI

This register is not defined in the EHCI specification. This register is only used by the DR module in device mode.

Offset 0x2_31B8 Access: Read only

	31–22	21–16	15–6	5–0
R	—	ETBR	—	ERBR
W				
Reset	All zeros			

Figure 16-28. Endpoint Status (ENDPTSTATUS)—DR-Only

Table 16-35. ENDPTSTATUS Register Field Descriptions

Bits	Name	Description
31–22	—	Reserved, should be cleared
21–16	ETBR	Endpoint transmit buffer ready. One bit for each endpoint indicates status of the respective endpoint buffer. This bit is set by the hardware as a response to receiving a command from a corresponding bit in the ENDPTPRIME register. There will always be a delay between setting a bit in the ENDPTPRIME register and endpoint indicating ready. This delay time varies based upon the current USB traffic and the number of bits set in the ENDPTPRIME register. Buffer ready is cleared by USB reset, by the USB DMA system, or through the ENDPTFLUSH register. ETBR[5] (bit 21 of the register) corresponds to endpoint 5. Note that these bits will be momentarily cleared by hardware during hardware endpoint re-priming operations when a dTD is retired, and the dQH is updated.
15–6	—	Reserved, should be cleared
5–0	ERBR	Endpoint receive buffer ready. One bit for each endpoint indicates status of the respective endpoint buffer. This bit is set by the hardware as a response to receiving a command from a corresponding bit in the ENDPTPRIME register. There will always be a delay between setting a bit in the ENDPTPRIME register and endpoint indicating ready. This delay time varies based upon the current USB traffic and the number of bits set in the ENDPTPRIME register. Buffer ready is cleared by USB reset, by the USB DMA system, or through the ENDPTFLUSH register. ERBR[5] corresponds to endpoint 5. Note that these bits will be momentarily cleared by hardware during hardware endpoint re-priming operations when a dTD is retired, and the dQH is updated.

16.3.2.23 Endpoint Complete Register (ENDPTCOMPLETE)—DR-Only, Non-EHCI

This register is not defined in the EHCI specification. This register is only used by the DR module in device mode.

Offset 0x2_31BC Access: w1c

	31–22	21–16	15–6	5–0
R	—	ETCE	—	ERCE
W		w1c		w1c
Reset	All zeros			

Figure 16-29. Endpoint Complete (ENDPTCOMPLETE)—DR-Only

Table 16-36. ENDPTCOMPLETE Register Field Descriptions

Bits	Name	Description
31–22	—	Reserved, should be cleared
21–16	ETCE	Endpoint transmit complete event. Each bit indicates a transmit event (IN/INTERRUPT) occurred and software should read the corresponding endpoint queue to determine the endpoint status. If the corresponding IOC bit is set in the Transfer Descriptor, then this bit will be set simultaneously with the USBINT. Writing a one will clear the corresponding bit in this register. ETCE[5] (bit 21 of the register) corresponds to endpoint 5.
15–6	—	Reserved, should be cleared
5–0	ERCE	Endpoint receive complete event. Each bit indicates a received event (OUT/SETUP) occurred and software should read the corresponding endpoint queue to determine the transfer status. If the corresponding IOC bit is set in the Transfer Descriptor, then this bit will be set simultaneously with the USBINT. Writing a one will clear the corresponding bit in this register. ERCE[5] corresponds to endpoint 5.

16.3.2.24 Endpoint Control Register 0 (ENDPTCTRL0)—DR-Only, Non-EHCI

This register is not defined in the EHCI specification. Every device will implement endpoint 0 as a control endpoint.

Offset 0x2_31C0 Access: Mixed

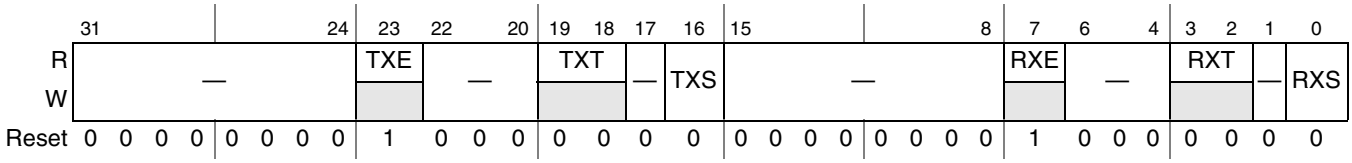


Figure 16-30. Endpoint Control 0 (ENDPTCTRL0)—DR-Only

Table 16-37. ENDPTCTRL0 Register Field Descriptions

Bits	Name	Description
31–24	—	Reserved, should be cleared.
23	TXE	TX Endpoint Enable. Endpoint zero is always enabled. 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. 1 Enabled.
6–4	—	Reserved, should be cleared.
3–2	RXT	RX Endpoint type. Endpoint zero is always a control endpoint (00).

Table 16-37. 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.

16.3.2.25 Endpoint Control Register *n* (ENDPTCTRL*n*)—DR-Only, Non-EHCI

These registers are not defined in the EHCI specification. There is an ENDPTCTRL*n* register of each endpoint in a device.

Offset 0x2_31C4 (ENDPTCTRL1), 0x2_31C8 (ENDPTCTRL2), 0x2_31CA (ENDPTCTRL3), 0x2_31D0 (ENDPTCTRL4), 0x2_31D4 (ENDPTCTRL5) Access: Read/write

	31–24	23	22	21	20	19–18	17	16	15–8	7	6	5	4	3–2	1	0
R W	—	TXE	TXR	TXI	—	TXT	TXD	TXS	—	RXE	RXR	RXI	—	RXT	RXD	RXS
Reset	All zeros															

Figure 16-31. Endpoint Control 1 to 5 (ENDPTCTRL*n*)—DR-Only

Table 16-38. ENDPTCTRLx Register Field Descriptions

Bits	Name	Description
31–24	—	Reserved, should be cleared
23	TXE	TX endpoint enable. 1 Enabled. 0 Disabled.
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. 1 PID Sequencing Disabled. 0 PID Sequencing Enabled.
20	—	Reserved, should be cleared
19–18	TXT	TX endpoint type. 00 Control. 01 Isochronous. 10 Bulk. 11 Interrupt. **Note:** When only one endpoint (RX or TX, but not both) of an endpoint pair is used, the unused endpoint should be configured as a bulk type endpoint.
17	TXD	TX endpoint data source. This bit should always be written as 0, which selects the Dual Port Memory/DMA Engine as the source.

Table 16-38. ENDPTCTRLx Register Field Descriptions (continued)

Bits	Name	Description
16	TXS	TX endpoint stall. This bit will be set automatically upon receipt of a SETUP request if this Endpoint is not configured as a Control Endpoint. It will be cleared automatically upon receipt of a SETUP request if this Endpoint is configured as a Control Endpoint. 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.
15–8	—	Reserved, should be cleared
7	RXE	RX endpoint enable. 1 Enabled. 0 Disabled.
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. **Note:** When only one endpoint (RX or TX, but not both) of an endpoint pair is used, the unused endpoint should be configured as a bulk type endpoint.
1	RXD	RX endpoint data sink. This bit should always be written as 0, which selects the Dual Port Memory/DMA Engine as the sink.
0	RXS	RX endpoint stall. This bit will be set automatically upon receipt of a SETUP request if this Endpoint is not configured as a Control Endpoint. It will be cleared automatically upon receipt a SETUP request if this Endpoint is configured as a Control Endpoint, 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.

16.3.2.26 SNOOP1 and SNOOP2—Non-EHCI

Note that these registers use big-endian byte ordering and are not defined in the EHCI specification. The SNOOP1and 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*n*[27–31]) are equal to 00000, snooping is always disabled on the CSB for all DMA transfers. When SNOOP*n*[27–31] is 01011 through 11110, the twenty upper bits (SNOOP*n*[0–19]) provide the starting base address for which transactions are snooped. These twenty bits are compared to the twenty upper bits of the address provided by the DMA block of the USB controller. When a match occurs, the five lower bits are decoded as shown below. This provides a snooping region of

4 Kbytes to 2 Gbytes within each starting base address that is programmed by the core. The SNOOP*n*[20–26] are not used.

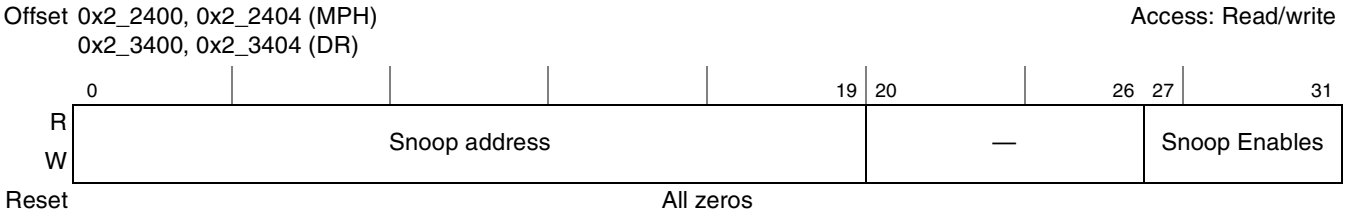


Figure 16-32. Snoop 1 and Snoop 2 (SNOOP*n*)

Table 16-39. SNOOP*n* Register Field Descriptions

Bits	Name	Description
0–19	Snoop address	The starting base address for which transactions are snooped.
20–26	—	Reserved, should be cleared
27–31	Snoop Enables	0x00 snooping disabled. 0x0B 4 Kbyte snoop range starting at the value defined by SNOOP*n*[0–19] 0x0C 8 Kbyte snoop range starting at the value defined by SNOOP*n*[0–18] 0x0D 16 Kbyte snoop range starting at the value defined by SNOOP*n*[0–17] 0x0E 32 Kbyte snoop range starting at the value defined by SNOOP*n*[0–16] 0x0F 64 Kbyte snoop range starting at the value defined by SNOOP*n*[0–15] 0x10 128 Kbyte snoop range starting at the value defined by SNOOP*n*[0–14] 0x11 256 Kbyte snoop range starting at the value defined by SNOOP*n*[0–13] 0x12 512 Kbyte snoop range starting at the value defined by SNOOP*n*[0–12] 0x13 1 Mbyte snoop range starting at the value defined by SNOOP*n*[0–11] 0x14 2 Mbyte snoop range starting at the value defined by SNOOP*n*[0–10] 0x15 4 Mbyte snoop range starting at the value defined by SNOOP*n*[0–9] 0x16 8 Mbyte snoop range starting at the value defined by SNOOP*n*[0–8] 0x17 16 Mbyte snoop range starting at the value defined by SNOOP*n*[0–7] 0x18 32 Mbyte snoop range starting at the value defined by SNOOP*n*[0–6] 0x19 64 M byte snoop range starting at the value defined by SNOOP*n*[0–5] 0x1A 31 Mbyte snoop range starting at the value defined by SNOOP*n*[0–4] 0x1B 256 Mbyte snoop range starting at the value defined by SNOOP*n*[0–3] 0x1C 512 Mbyte snoop range starting at the value defined by SNOOP*n*[0–2] 0x1D 1 Gbyte snoop range starting at the value defined by SNOOP*n*[0–1] 0x1E 2 Gbyte snoop range starting at the value defined by SNOOP*n*[0]

16.3.2.27 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 module'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 zero 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.

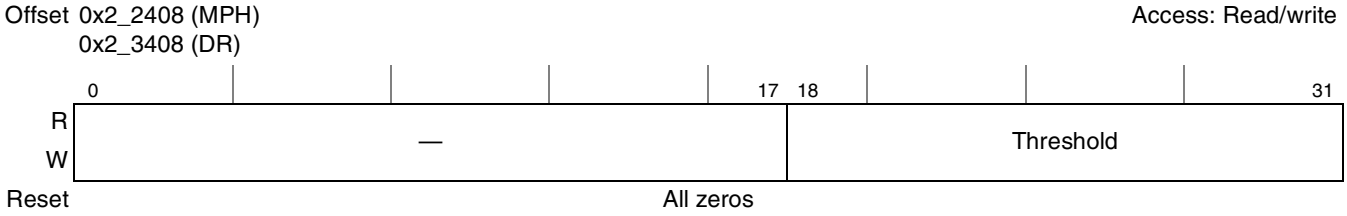


Figure 16-33. Age Count Threshold (AGE_CNT_THRESH)

Table 16-40. 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 the pri_lvl0 field of the PRI_CTRL register to 0.
- set the pri_lvl1 field of the PRI_CTRL register 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.

16.3.2.28 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.

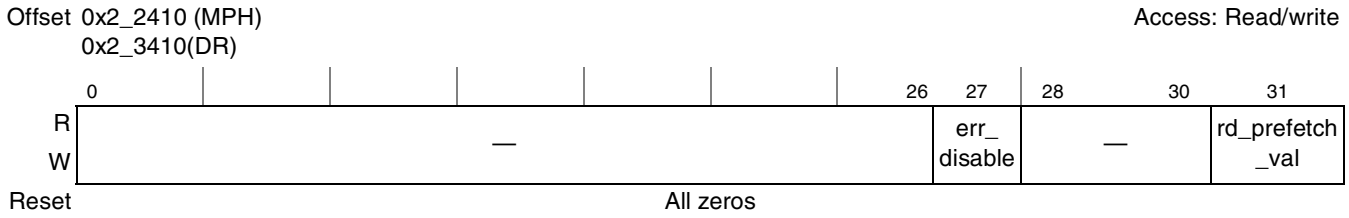


Figure 16-34. System Interface Control Register (SI_CTRL)

Table 16-41. SI_CTRL Register Field Descriptions

Bits	Name	Description
0–26	—	Reserved, should be cleared
27	err_disable	When this bit is HIGH, it causes the controller to ignore system bus errors. If LOW the controller responds according to the values set in the SEI bit of the USBSTS register and the SEE bit of the USBINT register. 1 Disable. 0 Enable.
28–30	—	Reserved, should be cleared
31	rd_prefetch_val	This register bit 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 set. Otherwise, this value should be left cleared. 1 64-byte fetch. 0 32-byte fetch.

16.3.2.29 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.

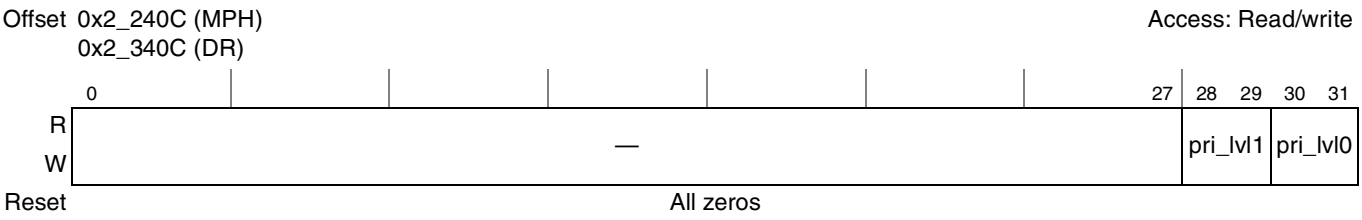


Figure 16-35. Priority Control (PRI_CTRL)

Table 16-42. PRI_CTRL Register Field Descriptions

Bits	Name	Description
0–27	—	Reserved, should be cleared
28–29	pri_lvl1	Priority level for priority state 1.
30–31	pri_lvl0	Priority level for priority state 0.

16.3.2.30 General Purpose Control Register (CONTROL)—Non-EHCI

Note that this register uses big-endian byte ordering and is not defined in the EHCI specification. The general purpose control (CONTROL) register is shown in Figure 16-54.

Offset 0x2_2500 (MPH)
0x2_3500 (DR)

Access: Read/write

	0	28	29	30	31
R W	—		USB_EN	ULPI _ INT1_EN	ULPI _ INT0_EN
Reset	All zeros				

Figure 16-36. Control (CONTROL)

Table 16-43. CONTROL Field Descriptions

Bits	Name	Description
0–28	—	Reserved, should be cleared
29	USB_EN	Used to enable the USB interface. In safe mode, all USB interface signals are put into input mode or driven inactive, except for SUSPEND_STP which is driven high. Also, the input signal 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 significantlly different times to complete power-on reset. 1 Normal operation. 0 Safe mode.
30	ULPI_INT1_EN (MPH-Only)	This bit is used to enable the ULPI low power wakeup interrupt from the PHY on Port 1 when the PHY is in low power mode only. 1 ULPI low power wakeup interrupt enabled. 0 ULPI low power wakeup interrupt disabled. This bit is reserved on the DR module.
31	ULPI_INT0_EN	This bit is used to enable the ULPI low power wakeup interrupt from the PHY on Port 0 when the PHY is in low power mode only. 1 ULPI low power wakeup interrupt enabled. 0 ULPI low power wakeup interrupt disabled.

16.4 Functional Description

Each module (MPH and DR) can be broken down into functional sub-blocks described below.

16.4.1 System Interface

The system interface block contains all the control and status registers that allow a core to interface to the module. These registers allow the processor to control the configuration of the module, ascertain the capabilities of and control the module's operation. It also has registers to control snoopability and priority of the DMA interface.

16.4.2 DMA Engine

Both modules contain local DMA engines. 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.

The DMA controllers must access both control information and packet data from system memory. The control information is contained in link list based queue structures. The DMA controllers have 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 speed devices. In device mode (DR module only), the data structures 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.

16.4.3 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 (DR module only), multiple FIFO channels are maintained for each of the active endpoints in the system.

In host mode, the MPH and DR modules use 512-byte TX buffers and a 512-byte RX buffers. For the DR module, device operation uses a single 512-byte RX buffer and a 512-byte TX buffer for each endpoint. The 512-byte buffers allow the modules to buffer a complete HS bulk packet. The MPH module is a single host controller with multiple ports and contains a single 512-byte TX buffer and a single 512-byte RX buffer.

16.4.4 PHY Interface

The MPH module interfaces to any ULPI compatible PHY, as well as FS/LS serial transceivers. The DR module interfaces to any UTMI or ULPI compatible PHY, and also to FS/LS serial transceivers. 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 it's associated resources.

Due to pincount limitations the modules only support certain combinations of PHY interfaces and USB functionality. Refer to Table 16-1 for more information.

16.5 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 via 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.

16.5.1 Periodic Frame List

Figure 16-37 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.

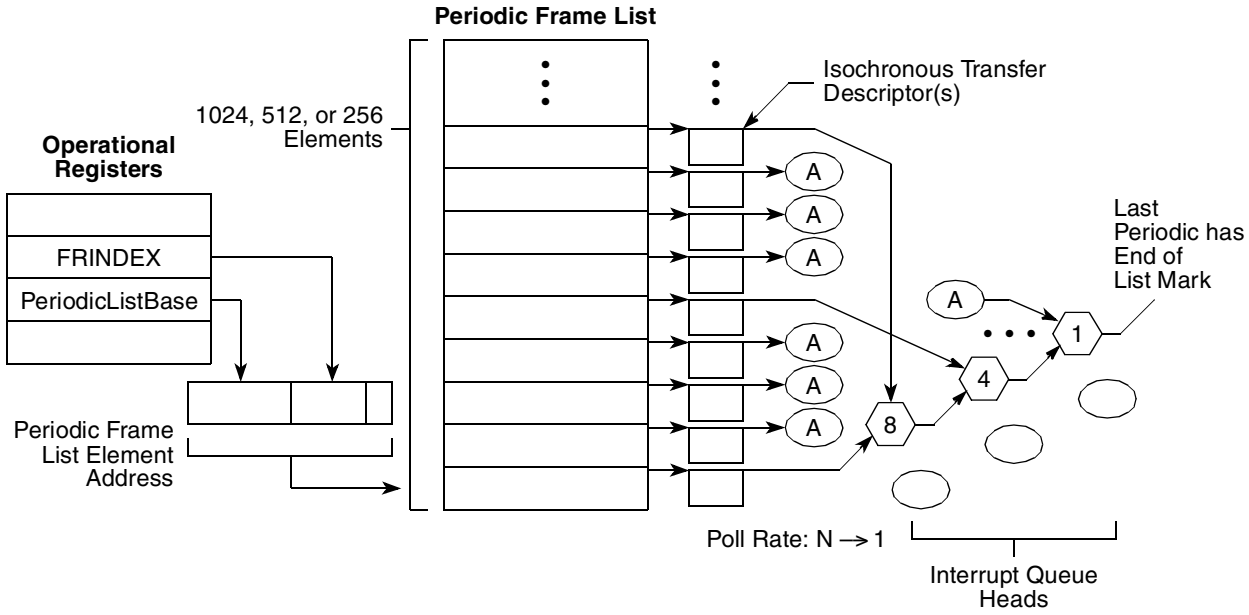


Figure 16-37. Periodic Schedule Organization

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 via the HCCPARAMS register. If non-programmable, the length is 1024 elements. If programmable, the length can be selected by system software as one of 256, 512, or 1024 elements. An implementation must support all three sizes. Programming the size (that is, the number of elements) is accomplished by system software writing the appropriate value into Frame List Size field in the USBCMD register.

Frame List Link pointers direct the host controller to the first work item in the frame's periodic schedule for the current micro-frame. The link pointers are aligned on DWord boundaries within the Frame List. Figure 16-38 shows the format for the Frame List Link Pointer.

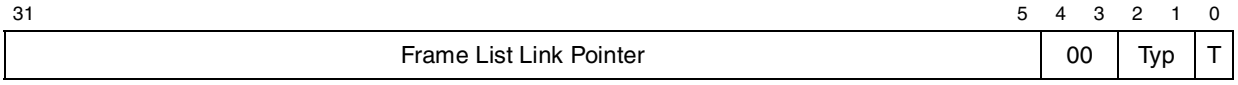


Figure 16-38. Frame List Link Pointer Format

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 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 will never use the value of the frame list pointer as a physical memory pointer. The Typ field is used to indicate the exact type of data structure being referenced by this pointer. The value encodings for the Typ field are given in Table 16-44.

Table 16-44. Typ Field Encodings

Typ	Description
00	Isochronous Transfer Descriptor
01	Queue Head
10	Split Transaction Isochronous Transfer Descriptor
11	Frame Span Traversal Node.

16.5.2 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.

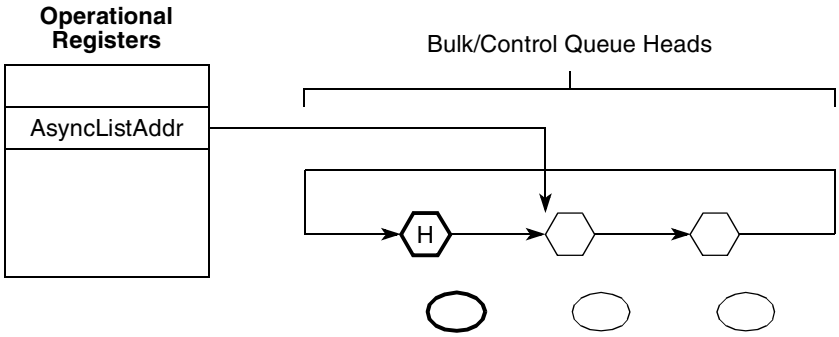


Figure 16-39. Asynchronous Schedule Organization

The Asynchronous list is a simple circular list of queue heads. The ASYNCLISTADDR register is simply pointer to the next queue head. This implements a pure round-robin service for all queue heads linked into the asynchronous list.

16.5.3 Isochronous (High-Speed) Transfer Descriptor (iTD)

The format of an isochronous transfer descriptor is illustrated in Figure 16-40. 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[1]	Transaction 0 Length[1]	ioc	PG[2]	Transaction 0 Offset[2]				0x04
Status[1]	Transaction 1 Length[1]	ioc	PG[2]	Transaction 1 Offset[2]				0x08
Status[1]	Transaction 2 Length[1]	ioc	PG[2]	Transaction 2 Offset[2]				0x0C
Status[1]	Transaction 3 Length[1]	ioc	PG[2]	Transaction 3 Offset[2]				0x10
Status[1]	Transaction 4 Length[1]	ioc	PG[2]	Transaction 4 Offset[2]				0x14
Status[1]	Transaction 5 Length[1]	ioc	PG[2]	Transaction 5 Offset[2]				0x18
Status[1]	Transaction 6 Length[1]	ioc	PG[2]	Transaction 6 Offset[2]				0x1C
Status[1]	Transaction 7 Length[1]	ioc	PG[2]	Transaction 7 Offset[2]				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 16-40. Isochronous Transaction Descriptor (iTD)

1 Host controller read/write; all others read-only.

2 These fields may be modified by the host controller if the I/O field indicates an OUT.

16.5.3.1 Next Link Pointer

The first DWord of an iTD is a pointer to the next schedule data structure.

Table 16-45. Next Schedule Element Pointer

Bit	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.

16.5.3.2 iTD Transaction Status and Control List

DWords 1 through 8 are eight slots of transaction control and status. Each transaction description includes:

- Status results field
- Transaction length (bytes to send for OUT transactions and bytes received for IN transactions).
- Buffer offset. The PG and Transaction *n* Offset fields are used with the buffer pointer list to construct the starting buffer address for the transaction.

The host controller uses the information in each transaction description plus the endpoint information contained in the first three DWords of the Buffer Page Pointer list, to execute a transaction on the USB.

Table 16-46. iTD Transaction Status and Control

Bit	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 sets this bit to zero 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 *n* Length	For an OUT, this field is the number of data bytes the host controller will send during the transaction. The host controller is not required to update this field to reflect the actual number of bytes transferred during the transfer. For an IN, the initial value of the endpoint to deliver. During the status update, the host controller writes back the field is the number of bytes the host expects the number of bytes successfully received. The value in this register is the actual byte count (for example, 0 zero length data, 1 one byte, 2 two bytes, etc.). The maximum value this field may contain is 0xC00 (3072).
15	ioc	Interrupt on complete. If this bit is set, it specifies that when this transaction completes, the Host Controller should issue an interrupt at the next interrupt threshold.
14–12	PG	These bits are set by software to indicate which of the buffer page pointers the offset field in this slot should be concatenated to produce the starting memory address for this transaction. The valid range of values for this field is 0 to 6.
11–0	Transaction *n* Offset	This field is a value that is an offset, expressed in bytes, from the beginning of a buffer. This field is concatenated onto the buffer page pointer indicated in the adjacent PG field to produce the starting buffer address for this transaction.

16.5.3.3 iTD Buffer Page Pointer List (Plus)

DWords 9-15 of an isochronous transaction descriptor are nominally page pointers (4K aligned) to the data buffer for this transfer descriptor. This data structure requires the associated data buffer to be contiguous (relative to virtual memory), but allows the physical memory pages to be non-contiguous. Seven page pointers are provided to support the expression of eight isochronous transfers. The seven pointers allow

for 3 (transactions) × 1024 (maximum packet size) × 8 (transaction records) = 24576 bytes to be moved with this data structure, regardless of the alignment offset of the first page.

Since each pointer is a 4K aligned page pointer, the least significant 12 bits in several of the page pointers are used for other purposes.

Table 16-47. Buffer Pointer Page 0 (Plus)

Bit	Name	Description
31–12	Buffer Pointer (Page 0)	This is a 4K aligned pointer to physical memory. Corresponds to memory address bits [31–12].
11–8	EndPt	This 4-bit field 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 16-48. iTD Buffer Pointer Page 1 (Plus)

Bit	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 (*wMaxPacketSize*). This field is used for high-bandwidth endpoints where more than one transaction is issued per transaction description (.for example, per micro-frame). This field is used with the *Multi* field to support high-bandwidth pipes. This field is also used for all IN transfers to detect packet babble. Software should not set a value larger than 1024 (400h). Any value larger yields undefined results.

Table 16-49. Buffer Pointer Page 2 (Plus)

Bit	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	This field is used to indicate to the host controller the number of transactions that should be executed per transaction description (for example, per micro-frame). The valid values are: 00 Reserved. A zero in this field yields undefined results. 01 One transaction to be issued for this endpoint per micro-frame 10 Two transactions to be issued for this endpoint per micro-frame 11 Three transactions to be issued for this endpoint per micro-frame

Table 16-50. Buffer Pointer Page 3-6

Bit	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.

16.5.4 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.

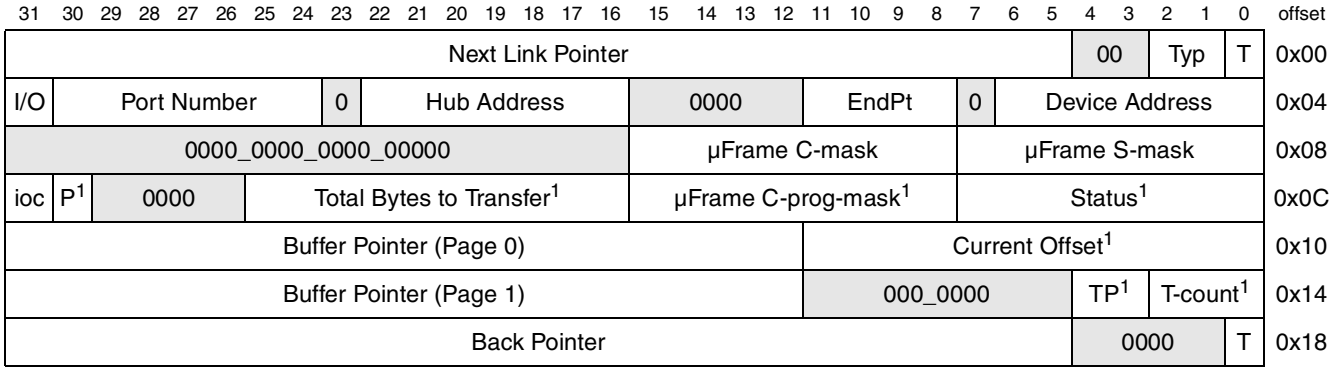


Figure 16-41. Split-Transaction Isochronous Transaction Descriptor (siTD)

1 Host controller read/write; all others read-only.

16.5.4.1 Next Link Pointer

DWord0 of a siTD is a pointer to the next schedule data structure.

Table 16-51. Next Link Pointer

Bit	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	This field 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.

16.5.4.2 siTD Endpoint Capabilities/Characteristics

DWords 1 and 2 specify static information about the full-speed endpoint, the addressing of the parent Companion Controller, and micro-frame scheduling control.

Table 16-52. Endpoint and Transaction Translator Characteristics

Bit	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. This 4-bit field 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	This field selects the specific device serving as the data source or sink.

Table 16-53. Micro-frame Schedule Control

Bit	Name	Description
31–16	—	Reserved, should be cleared. This field reserved for future use. It should be cleared.
15–8	µFrame C-mask	Split Completion Mask. This field (along with the Active and SplitX- state fields in the Status byte) is used to determine during which micro-frames the host controller should execute complete-split transactions. When the criteria for using this field is met, an all zeros value has undefined behavior. The host controller uses the value of the three low-order bits of the FRINDEX register to index into this bit field. If the FRINDEX register value indexes to a position where the µFrame C-Mask field is a one, then this siTD is a candidate for transaction execution. There may be more than one bit in this mask set.
7–0	µFrame S-mask	Split Start Mask. This field (along with the Active and SplitX-state fields in the Status byte) is used to determine during which micro-frames the host controller should execute start-split transactions. The host controller uses the value of the three low-order bits of the FRINDEX register to index into this bit field. If the FRINDEX register value indexes to a position where the µFrame S-mask field is a one, then this siTD is a candidate for transaction execution. An all zeros value in this field, in combination with existing periodic frame list has undefined results.

16.5.4.3 siTD Transfer State

DWords 3-6 are used to manage the state of the transfer.

Table 16-54. siTD Transfer Status and Control

Bit	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. Used to indicate 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:	
		Status Bit	**Definition**
		7	Active. Set by software to enable the execution of an isochronous split transaction by the Host Controller.
		6	ERR. Set by the Host Controller when an ERR response is received from the Companion Controller.
		5	Data Buffer Error. Set by the Host Controller during status update to indicate that the Host Controller is unable to keep up with the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (under run). In the case of an under run, the Host Controller will transmit an incorrect CRC (thus invalidating the data at the endpoint). If an overrun condition occurs, no action is necessary.
		4	Babble Detected. Set by the Host Controller during status update when” babble” is detected during the transaction generated by this descriptor.
		3	Transaction Error (XactErr). Set by the Host Controller during status update in the case where the host did not receive a valid response from the device (Time-out, CRC, Bad PID, etc.). This bit will only be set for IN transactions.
		2	Missed Micro-Frame. The host controller detected that a host-induced hold- off caused the host controller to miss a required complete-split transaction.
		1	Split Transaction State (SplitXstate). The bit encodings are: 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.

16.5.4.4 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 16-55. siTD Buffer Pointer Page 0 (Plus)

Bit	Name	Description
31–12	Buffer Pointer (Page 0)	Bits [31–12] is a 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 16-56. siTD Buffer Pointer Page 1 (Plus)

Bit	Name	Description
31–12	Buffer Pointer (Page 1)	Bits [31–12] is a 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.

16.5.4.5 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 16-57. siTD Back Link Pointer

Bit	Name	Description
31–5	Back Pointer	This field is a physical memory pointer to a siTD.
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

16.5.5 Queue Element Transfer Descriptor (qTD)

This data structure is only used with a queue head. This data structure is used for one or more USB transactions. This data structure is used to transfer up to 20480 (5×4096) bytes. The structure contains two structure pointers used for queue advancement, a DWord of transfer state, and a five-element array of data buffer pointers. This structure is 32 bytes (or one 32-byte cache line). This data structure must be physically contiguous.

The buffer associated with this transfer must be virtually contiguous. The buffer may start on any byte boundary. A separate buffer pointer list element must be used for each physical page in the buffer, regardless of whether the buffer is physically contiguous.

Host controller updates (host controller writes) to stand-alone qTDs only occur during transfer retirement. References in the following bit field definitions of updates to the qTD are to the qTD portion of a queue head.

31–16	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[1]	Total Bytes to Transfer[1]	ioc	C_Page[1]	Cerr[1]	PID Code		Status[1]			0x08
Buffer Pointer (Page 0)				Current Offset[1]						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 16-42. Queue Element Transfer Descriptor (qTD)

[1] Host controller read/write; all others read-only.

Queue Element Transfer Descriptors must be aligned on 32-byte boundaries.

16.5.5.1 Next qTD Pointer

The first DWord of an element transfer descriptor is a pointer to another transfer element descriptor.

Table 16-58. qTD Next Element Transfer Pointer (DWord 0)

Bit	Name	Description
31–5	Next qTD Pointer	This field contains the physical memory address of the next qTD to be processed. 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. This bit 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.

16.5.5.2 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 16-59. qTD Alternate Next Element Transfer Pointer (DWord 1)

Bit	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. This bit 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.

16.5.5.3 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 16-60 reference fields defined in the queue head. See Section 16.5.6, “Queue Head,” for more information on these fields.

Table 16-60. qTD Token (DWord 2)

Bit	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, it specifies that when this qTD is completed, the Host Controller should issue an interrupt at the next interrupt threshold.
14–12	C_Page	Current Page. 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 16-60. qTD Token (DWord 2) (continued)

<table>
<tr><th>Bit</th><th>Name</th><th colspan="2">Description</th></tr>
<tr><td>11–10</td><td>Cerr</td><td colspan="2">Error Counter. This field is a 2-bit down counter that keeps track of the number of consecutive errors detected while executing this qTD. If this field is programmed with a non-zero value during set-up, the host controller decrements the count and writes it back to the qTD if the transaction fails. If the counter counts from one to zero, the host controller marks the qTD inactive, sets the Halted bit to a one, and error status bit for the error that caused Cerr to decrement to zero. An interrupt will be generated if the USB Error Interrupt Enable bit in the USBINTR register is set. If the host controller driver (HCD) software programs this field to zero during set-up, the host controller will not count errors for this qTD and there will be no limit on the retries of this qTD. Note that write-backs of intermediate execution state are to the queue head overlay area, not the qTD.</td></tr>
<tr><td></td><td></td><td>Error</td><td>Decrement Counter</td></tr>
<tr><td></td><td></td><td>Transaction Error</td><td>Yes</td></tr>
<tr><td></td><td></td><td>Data Buffer Error</td><td>No· Data buffer errors are host problems. They don't count against the device's retries. Note that software must not program Cerr to a value of zero when the EPS field is programmed with a value indicating a Full- or Low-speed device. This combination could result in undefined behavior.</td></tr>
<tr><td></td><td></td><td>Stalled</td><td>No· Detection of Babble or Stall automatically halts the queue head. Thus, count is not decremented</td></tr>
<tr><td></td><td></td><td>Babble Detected</td><td>No· Detection of Babble or Stall automatically halts the queue head. Thus, count is not decremented</td></tr>
<tr><td></td><td></td><td>No Error</td><td>No· If the EPS field indicates a HS device or the queue head is in the Asynchronous Schedule (and PIDCode indicates an IN or OUT) and a bus transaction completes and the host controller does not detect a transaction error, then the host controller should reset Cerr to extend the total number of errors for this transaction. For example, Cerr should be reset with maximum value (0b11) on each successful completion of a transaction. The host controller must never reset this field if the value at the start of the transaction is 0b00.</td></tr>
<tr><td>9–8</td><td>PID Code</td><td colspan="2">This field is an encoding of the token, which should be used for transactions associated with this transfer descriptor. Encodings are:
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. µFrame S-mask field in the queue head is non-zero.)
11 Reserved</td></tr>
</table>

Table 16-60. qTD Token (DWord 2) (continued)

<table>
<tr><th>Bit</th><th>Name</th><th colspan="2">Description</th></tr>
<tr><td>7–0</td><td>Status</td><td colspan="2">This field is used by the host controller to communicate individual command execution states back to the host controller driver (HCD) software. This field contains the status of the last transaction performed on this qTD. The bit encodings are:</td></tr>
<tr><td></td><td></td><td>Bit</td><td>Status Field Description</td></tr>
<tr><td></td><td></td><td>7</td><td>Active. Set by software to enable the execution of transactions by the host controller.</td></tr>
<tr><td></td><td></td><td>6</td><td>Halted. Set by the host controller during status updates to indicate that a serious error has occurred at the device/endpoint addressed by this qTD. This can be caused by babble, the error counter counting down to zero, or reception of the STALL handshake from the device during a transaction. Any time that a transaction results in the Halted bit being set, the Active bit is also cleared.</td></tr>
<tr><td></td><td></td><td>5</td><td>Data Buffer Error. Set by the host controller during status update to indicate that the host controller is unable to keep up with the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (under run). If an overrun condition occurs, the host controller will force a time-out condition on the USB, invalidating the transaction at the source. If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.</td></tr>
<tr><td></td><td></td><td>4</td><td>Babble Detected. Set by the host controller during status update when babble is detected during the transaction. In addition to setting this bit, the host controller also sets the Halted bit to a one. Since babble is considered a fatal error for the transfer, setting the Halted bit to a one insures that no more transactions occur because of this descriptor.</td></tr>
<tr><td></td><td></td><td>3</td><td>Transaction Error (XactErr). Set by the host controller during status update in the case where the host did not receive a valid response from the device (time-out, CRC, bad PID). If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.</td></tr>
<tr><td></td><td></td><td>2</td><td>Missed Micro-Frame. This bit is ignored unless the QH[EPS] field indicates a full- or low-speed endpoint and the queue head is in the periodic list. This bit is set when the host controller detected that a host-induced hold-off caused the host controller to miss a required complete-split transaction. If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.</td></tr>
</table>

Table 16-60. qTD Token (DWord 2) (continued)

Bit	Name	Description	
7–0 (cont.)	Status (cont.)	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.

16.5.5.4 qTD Buffer Page Pointer List

The last five DWords of a queue element transfer descriptor is an array of physical memory address pointers. These pointers reference the individual pages of a data buffer.

System software initializes Current Offset field to the starting offset into the current page, where current page is selected via the value in the C_Page field.

Table 16-61. qTD Buffer Pointer

Bit	Name	Description
31–12	Buffer Pointer (page *n*)	Each element in the list is a 4K page aligned physical memory address. The lower 12 bits in each pointer are reserved (except for the first one), as each memory pointer must reference the start of a 4K page. The field C_Page specifies the current active pointer. When the transfer element descriptor is fetched, the starting buffer address is selected using C_Page (similar to an array index to select an array element). If a transaction spans a 4K buffer boundary, the host controller must detect the page-span boundary in the data stream, increment C_Page and advance to the next buffer pointer in the list, and conclude the transaction via the new buffer pointer.
11–0	Current Offset (Page 0)/ — (Pages 1-4)	This field is 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.

16.5.6 Queue Head

Figure 16-43 shows the queue head structure.

31–27	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	dtc	EPS	EndPt		I	Device Address				0x04[1]
Mult		Port Number			Hub Addr		μFrame C-mask					μFrame S-mask					0x08[1]
Current qTD Pointer[2]														00000			0x0C
Next qTD Pointer[2]														0000		T[2]	0x10[3]
Alternate Next qTD Pointer[2]														NakCnt[2]		T[2]	0x14[3,4]
dt[1]	Total Bytes to Transfer[2]						ioc[2]	C_Page[2]		Cerr[2]	PID Code[2]		Status[2]				0x18[3,4]
Buffer Pointer (Page 0)[2]										Current Offset[2]							0x1C[3,4]
Buffer Pointer (Page 1)[2]										0000		C-prog-mask[2]					0x20[3,4]
Buffer Pointer (Page 2)[2]										S-bytes[2]					FrameTag[2]		0x24[3,4]
Buffer Pointer (Page 3)[2]										0000_0000_0000							0x28[3]
Buffer Pointer (Page 4)[2]										0000_0000_0000							0x2C[3]

Figure 16-43. Queue Head Layout

1 Offsets 0x04 through 0x0B contain the static endpoint state.
2 Host controller read/write; all others read-only.
3 Offsets 0x10 through 0x2F contain the transfer overlay.
4 Offsets 0x14 through 0x27 contain the transfer results.

16.5.6.1 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 16-62. Queue Head DWord 0

Bit	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 16-62. Queue Head DWord 0 (continued)

Bit	Name	Description
2–1	Typ	This field 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.

16.5.6.2 Endpoint Capabilities/Characteristics

The second and third DWords of a Queue Head specifies 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 including addressing, maximum packet size, and endpoint speed.
- Endpoint Capabilities. These are adjustable parameters of the endpoint. They effect how the endpoint data stream is managed by the host controller.
- Split Transaction Characteristics. This data structure is used to manage full- and low-speed data streams for bulk, control, and interrupt via split transactions to USB2.0 Hub Transaction Translator. There are additional fields used for addressing the hub and scheduling the protocol transactions (for periodic).

The host controller must not modify the bits in this region.

Table 16-63. Endpoint Characteristics: Queue Head DWord 1

Bit	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 16-63. Endpoint Characteristics: Queue Head DWord 1 (continued)

Bit	Name	Description
14	dtc	Data Toggle Control (DTC). This bit 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 (12Mbs) 01 Low-Speed (1.5Mbs) 10 High-Speed (480 Mb/s) 11 Reserved. This field must not be modified by the host controller.
11–8	EndPt	Endpoint Number. This 4-bit field 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 to a one 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	This field selects the specific device serving as the data source or sink.

Table 16-64. Endpoint Capabilities: Queue Head DWord 2

Bit	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. A zero in this field yields undefined results. 01 One transaction to be issued for this endpoint per micro-frame 10 Two transactions to be issued for this endpoint per micro-frame 11 Three transactions to be issued for this endpoint per micro-frame
29–23	Port Number.	This field is ignored by the host controller unless the EPS field indicates a full- or low-speed device. The value is the port number identifier on the USB 2.0 hub (for hub at device address Hub Addr below), below which the full- or low-speed device associated with this endpoint is attached. This information is used in the split-transaction protocol.
22–16	Hub Addr	This field is ignored by the host controller unless the EPS field indicates a full-or low-speed device. The value is the USB device address of the USB 2.0 hub below which the full- or low-speed device associated with this endpoint is attached. This field is used in the split-transaction protocol.

Table 16-64. Endpoint Capabilities: Queue Head DWord 2 (continued)

Bit	Name	Description
15–8	µFrame C-mask	This field is ignored by the host controller unless the EPS field indicates this device is a low- or full-speed device and this queue head is in the periodic list. This field (along with the Active and SplitX-state fields) is used to determine during which micro-frames the host controller should execute a complete-split transaction. When the criteria for using this field are met, a zero value in this field has undefined behavior. This field is used by the host controller to match against the three low-order bits of the FRINDEX register. If the FRINDEX register bits decode to a position where the µFrame C- mask field is a one, then this queue head is a candidate for transaction execution. There may be more than one bit in this mask set.
7–0	µFrame S-mask	Interrupt Schedule Mask. This field is used for all endpoint speeds. Software should set this field to a zero when the queue head is on the asynchronous schedule. A non-zero value in this field indicates an interrupt endpoint. The host controller uses the value of the three low-order bits of the FRINDEX register as an index into a bit position in this bit vector. If the µFrame S-mask field has a one at the indexed bit position then this queue head is a candidate for transaction execution. If the EPS field indicates the endpoint is a high-speed endpoint, then the transaction executed is determined by the PID_Code field contained in the execution area. This field is also used to support split transaction types: Interrupt (IN/OUT). This condition is true when this field is non-zero and the EPS field indicates this is either a full- or low-speed device. A zero value in this field, in combination with existing in the periodic frame list has undefined results.

16.5.6.3 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 16-65. Current qTD Link Pointer

Bit	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 16-66. Host-Controller Rules for Bits in Overlay (DWords 5, 6, 8 and 9)

DWord	QH Offset	Bit	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. This two-bit field is 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. This field is 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. This field is used to keep 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. This field is initialized to zero during any overlay. This field is used to track the progress of an interrupt split-transaction.

16.5.7 Periodic Frame Span Traversal Node (FSTN)

This data structure is to be used only for managing Full- and Low-speed transactions that span a Host-frame boundary. Software must not use an FSTN in the Asynchronous Schedule. An FSTN in the Asynchronous schedule results in undefined behavior. Software must not use the FSTN feature with a host controller whose HCIVERSION register indicates a revision implementation below 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.

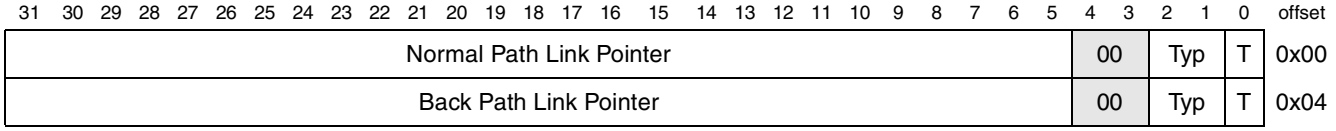


Figure 16-44. Frame Span Traversal Node Structure

16.5.7.1 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 16-67. FTSN Normal Path Pointer

Bit	Name	Description
31–5	NPLP	Normal Path 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 0s.
2–1	Typ	This field indicates to the host controller whether the item referenced is a iTD/siTD, a QH or an 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.

16.5.7.2 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 16-68. FSTN Back Path Link Pointer

Bit	Name	Description
31–5	BPLP	Back Path Link Pointer. This field 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.

16.6 Host Operations

The general operational model is for the USB modules in host mode is defined by the Enhanced Host Controller Interface (EHCI) Specification. The EHCI specification describes the register-level interface for a host controller for the USB Revision 2.0. It includes a description of the hardware/software interface

between system software and host controller hardware. Information concerning the initialization of the USB modules is included in the following section; however, the full details of the EHCI specification are beyond the scope of this document.

16.6.1 Host Controller Initialization

After initial power-on or HCReset (hardware or via HCReset bit in the USBCMD register), all of the operational registers will be at their default values, as illustrated in Table 25. After a hardware reset, only the operational registers not contained in the Auxiliary power well will be at their default values.

Table 16-69. Default Values of Operational Register Space

Operational Register	Default Value (after Reset)
USBCMD	0x0008_0000 (0x0008_0B00 if Asynchronous Schedule Park Capability is set)
USBSTS	0x0000_1000
USBINTR	0x0000_0000
FRINDEX	0x0000_0000
CTRLDSSEGMENT	0x0000_0000
PERIODICLISTBASE	Undefined
ASYNCLISTADDR	Undefined
CONFIGFLAG	0x0000_0000
PORTSC	0x0000_2000 (w/PPC set); 0x0000_3000 (w/PPC cleared)

In order to initialize the host controller, software should perform the following steps

1. Optionally set Streaming Disable in the USBMODE register.
2. Optionally modify the BURSTSIZE register.
3. Program the PTS field of the PORTSCx register if using a non-ULPI PHY.
4. Set the USB_EN bit in the CONTROL register.
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 via setting the *Run/Stop* bit.

At this point, the host controller 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 the *Asynchronous Schedule Enable* bit in the USBCMD register. In order to communicate with devices via the periodic schedule, system software must enable the periodic schedule by writing a one to the *Periodic Schedule Enable* bit in the USBCMD register. 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.

16.6.2 Power Port

The Port Power Control (PPC) bit in the HCSPARAMS register indicates whether the USB 2.0 host controller has port power control. When the PPC bit is a one, then 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.

16.6.3 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. Each 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 MPH and DR logic. The over-current condition effects the following bits in the PORTSC register on the EHCI port:

- Over-current Active bit is set. When the over-current condition goes away, the Over-current Active bit will transition from a one to a zero.
- Over-current Change bit is set. On every transition of the Over-current Active bit the controller will set the Over-current Change bit to a one. Software sets the Over-current Change bit to a zero by writing a one to this bit.
- Port Enabled/Disabled bit is cleared. When this change bit gets set, then the Port Change Detect bit in the USBSTS register 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 the Over-current Change bit transitions from a zero to a one, the controller also sets the Port Change Detect bit in the USBSTS register to a one. In addition, if the Port Change Interrupt Enable bit in the USBINTR register is a one, then the controller issues an interrupt to the system. Refer to Table 16-70 for summary of behavior for over-current detection when the controller is halted (suspended from a device component point of view).

16.6.4 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 via software initiation. Other control mechanisms are provided to parameterize the host controller's response (or sensitivity) to external resume events. In this discussion, host-initiated, or software-initiated resumes are called Resume Events/Actions; bus-initiated resume events are called wake-up events. The classes of wakeup events are:

- Remote-wakeup enabled device asserts resume signaling. In similar kind to USB 2.0 hubs, when in host mode the host controller responds to explicit device resume signaling and wake up the system (if necessary).
- Port connect and disconnect and over-current events. Sensitivity to these events can be turned on or off by using the port control bits in the PORTSC register.

Selective suspend is a feature supported by the PORTSC register. It is used to place specific ports into a suspend mode. This feature is used as a functional component for implementing the appropriate power management policy implemented in a particular operating system. When system software intends to suspend the bus, it should suspend the enabled port, then shut off the controller by setting the Run/Stop bit in the USBCMD register to a zero.

When a wake event occurs the system will resume operation and system software must set the Run/Stop bit to a one and resume the suspended port.

16.6.4.1 Port Suspend/Resume

System software places the USB into suspend mode by writing a one into the appropriate PORTSC Suspend bit. Software must only set the Suspend bit when the port is in the enabled state (Port Enabled bit is a one).

The host controller may evaluate the Suspend bit immediately or wait until a micro-frame or frame boundary occurs. If evaluated immediately, the port is not suspended until the current transaction (if one is executing) completes. Therefore, there may be several micro-frames of activity on the port until the host controller evaluates the Suspend bit. The host controller must evaluate the Suspend bit at least every frame boundary.

System software can initiate a resume on the suspended port by writing a one to the Force Port Resume bit. Software should not attempt to resume a port unless the port reports that it is in the suspended state. If system software sets the Force Port Resume bit 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 via the Force Port Resume bit. When Force Port Resume bit is set, the host controller sends resume signaling down the port. System software times the duration of the resume (nominally 20 milliseconds) then clears the Force Port Resume bit. When the host controller receives the write to transition Force Port Resume to zero, it completes the resume sequence as defined in the USB specification, and clears both the Force Port Resume and Suspend bits. Software-initiated port resumes do not affect the Port Change Detect bit in the USBSTS register nor do they cause an interrupt if the Port Change Interrupt Enable bit in the

USBINTR register is a one. When a wake event occurs on a suspended port, the resume signaling is detected by the port and the resume is reflected downstream within 100 µsec. The port's Force Port Resume bit is set and the Port Change Detect bit in the USBSTS register is set. If the Port Change Interrupt Enable bit in the USBINTR register 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 the Force Port Resume bit in the port. The host controller receives the write of zero to Force Port Resume, terminates the resume sequence and clears the Force Port Resume and Suspend port bits. Software can determine that the port is enabled (not suspended) by sampling the PORTSC register and observing that the Suspend and Force Port Resume bits are zero. Software must ensure that the host controller is running (that is, HCHalted bit in the USBSTS register is a zero), before terminating a resume by clearing the port's Force Port Resume bit. If HCHalted is a one when Force Port Resume 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 16-70 summarizes the wake-up events. Whenever a resume event is detected, the Port Change Detect bit in the USBSTS register is set. If the Port Change Interrupt Enable bit is a one in the USBINTR register, the host controller also generates an interrupt on the resume event. Software acknowledges the resume event interrupt by clearing the Port Change Detect status bit in the USBSTS register.

Table 16-70. 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. Force Port Resume status bit in PORTSC register is set. Port Change Detect bit in USBSTS register is set.	[1], [2]	[2]
Port is enabled, disabled or suspended, and the port's WKDSCNNT_E bit is set. A disconnect is detected.	Depending in the initial port state, the PORTSC Connect and Enable status bits are cleared, and the Connect Change status bit is set. Port Change Detect bit in the USBSTS register is set.	[1], [2]	[2]
Port is enabled, disabled or suspended, and the port's WKDSCNNT_E bit is cleared. A disconnect is detected.	Depending on the initial port state, the PORTSC Connect and Enable status bits are cleared, and the Connect Change status bit is set. Port Change Detect bit in the USBSTS register 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 and Connect Status Change bits are set. Port Change Detect bit in the USBSTS register 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 and Connect Status Change bits are set. Port Change Detect bit in the USBSTS register is set.	[1], [3]	[3]

Table 16-70. Behavior During Wake-up Events (continued)

Port Status and Signaling Type	Signaled Port Response	Device State	
		D0	not D0
Port is connected and the port's WKOC_E bit is a one. An over-current condition occurs.	PORTSC Over-current Active, Over-current Change bits are set. If Port Enable/Disable bit is a one, it is cleared. Port Change Detect bit in the USBSTS register 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, Over-current Change bits are set. If Port Enable/Disable bit is a one, it is cleared. Port Change Detect bit in the USBSTS register is set.	[1], [3]	[3]

[1] Hardware interrupt issued if Port Change Interrupt Enable bit in the USBINTR register is set.
[2] PME# asserted if enabled (Note: PME Status must always be set).
[3] PME# not asserted.

16.6.5 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 16.3.2.6, “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 16.5, “Host Data Structures.” In each micro-frame, if the periodic schedule is enabled (see) then the host controller must execute from the periodic schedule before executing from the asynchronous schedule. It will only execute from the asynchronous schedule after it encounters the end of the periodic schedule. The host controller traverses the periodic schedule by constructing an array offset reference from the PERIODICLISTBASE and the FRINDEX registers (see Figure 16-45). It fetches the element and begins traversing the graph of linked schedule data structures.

The end of the periodic schedule is identified by a next link pointer of a schedule data structure having its T-bit set. When the host controller encounters a T-Bit set during a horizontal traversal of the periodic list, it interprets this as an End-Of-Periodic-List mark. This causes the host controller to cease working on the periodic schedule and transitions immediately to traversing the asynchronous schedule. Once this transition is made, the host controller executes from the asynchronous schedule until the end of the micro-frame.

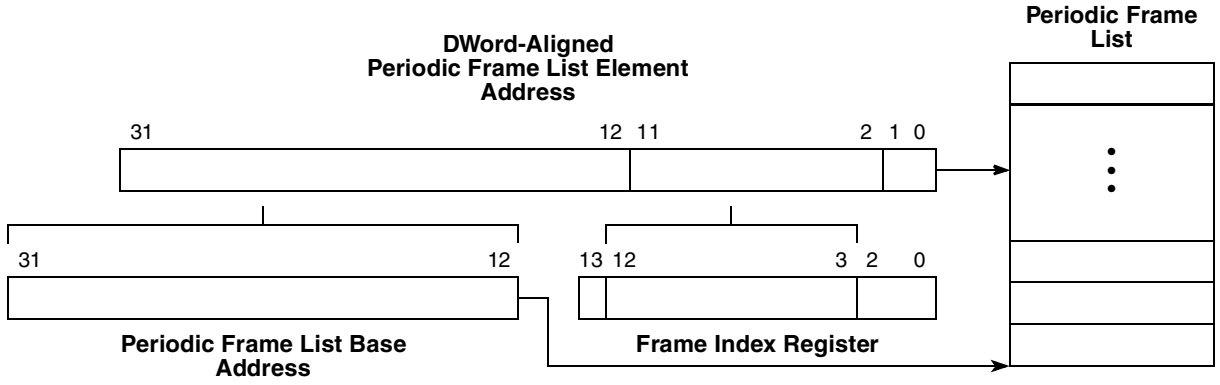


Figure 16-45. Derivation of Pointer into Frame List Array

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 16-46.

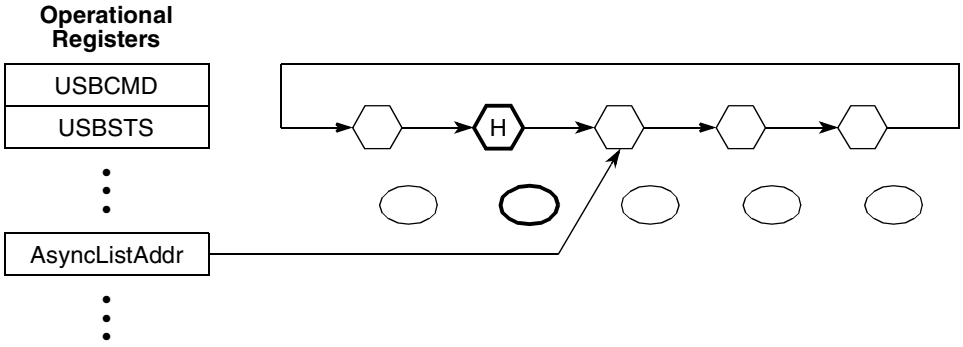


Figure 16-46. General Format of Asynchronous Schedule List

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.

16.6.6 Periodic Schedule Frame Boundaries vs. Bus Frame Boundaries

The USB Specification Revision 2.0 requires that the frame boundaries (SOF frame number changes) of the high-speed bus and the full- and low-speed bus(es) below USB 2.0 hubs be strictly aligned. Super-imposed on this requirement is that USB 2.0 hubs manage full- and low-speed transactions via a micro-frame pipeline (see start- (SS) and complete- (CS) splits illustrated in Figure 16-47). 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.

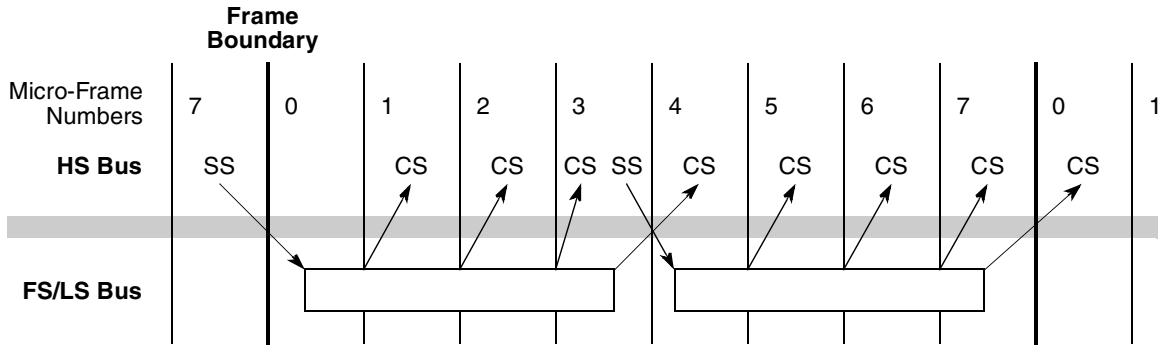


Figure 16-47. Frame Boundary Relationship between HS Bus and FS/LS Bus

The simple projection, as Figure 16-47 illustrates, introduces frame-boundary wrap conditions for scheduling on both the beginning and end of a frame. In order to reduce the complexity for hardware and software, the host controller is required to implement a one micro-frame phase shift for its view of frame boundaries. The phase shift eliminates the beginning of frame and frame-wrap scheduling boundary conditions.

The implementation of this phase shift requires that the host controller use one register value for accessing the periodic frame list and another value for the frame number value included in the SOF token. These two values are separate, but tightly coupled. The periodic frame list is accessed via the Frame List Index Register (FRINDEX). Bits FRINDEX[2–0], represent the micro-frame number. The SOF value is coupled to the value of FRINDEX[13–3]. Both FRINDEX[13–3] and the SOF value are incremented based on FRINDEX[2–0]. It is required that the SOF value be delayed from the FRINDEX value by one micro-frame. The one micro-frame delay yields a host controller periodic schedule and bus frame boundary relationship as illustrated in Figure 16-48. 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 16-48 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.

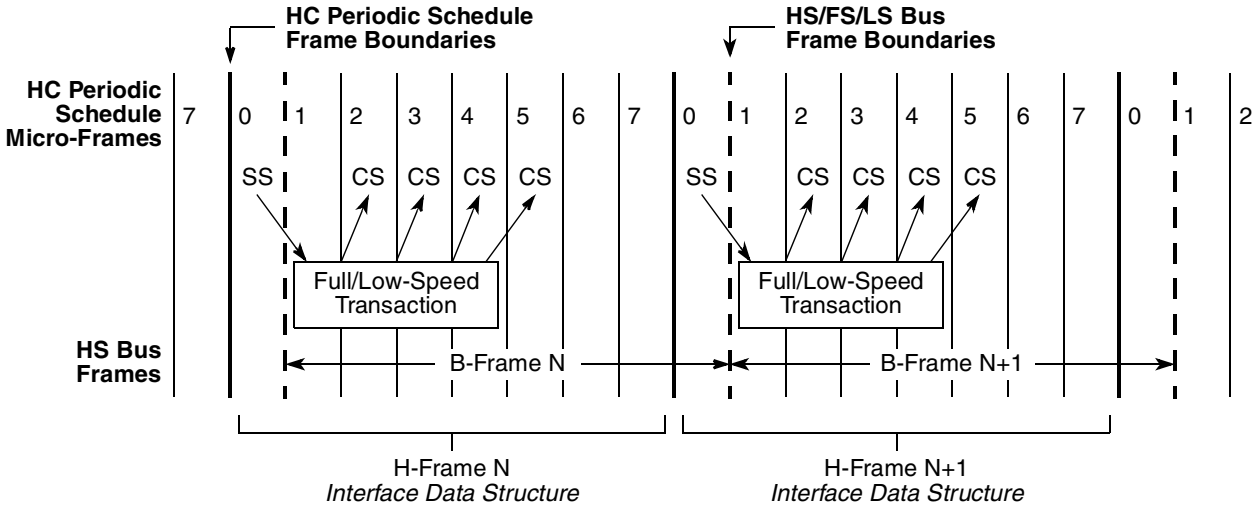


Figure 16-48. Relationship of Periodic Schedule Frame Boundaries to Bus Frame Boundaries

H-Frame boundaries for the host controller correspond to increments of FRINDEX[13–3]. Micro-frame numbers for the H-Frame are tracked by FRINDEX[2–0]. B-Frame boundaries are visible on the high-speed bus via changes in the SOF token's frame number. Micro-frame numbers on the high-speed bus are only derived from the SOF token's frame number (that is, the high-speed bus will see eight SOFs with the same frame number value). H-Frames and B-Frames have the fixed relationship (that is, B-Frames lag H-Frames by one micro-frame time) illustrated in Figure 16-48. 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 16.3.2.4, “Frame Index Register (FRINDEX),” the SOF Value can be implemented as a shadow register (in this example, called SOFV), which lags the FRINDEX register bits [13–3] by one micro-frame count. Table 16-71 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 16.3.2.4, “Frame Index Register (FRINDEX),” provides the requirements that software should adhere when writing a new value in FRINDEX.

Table 16-71. 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

16.6.7 Periodic Schedule

The periodic schedule traversal is enabled or disabled via the Periodic Schedule Enable bit in the USBCMD register. If the Periodic Schedule Enable bit is cleared, then the host controller simply does not try to access the periodic frame list via the PERIODICLISTBASE register. Likewise, when the Periodic Schedule Enable bit 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 the Periodic Schedule Enable immediately. In order to eliminate conflicts with split transactions, the host controller evaluates the Periodic Schedule Enable bit only when FRINDEX[2–0] is zero. System software must not disable the periodic schedule if the schedule contains an active split transaction work item that spans the 0b000 micro-frame. These work items must be removed from the schedule before the Periodic Schedule Enable bit is cleared. The Periodic Schedule Status bit in the USBSTS register indicates status of the periodic schedule. System software enables (or disables) the periodic schedule by setting (or clearing) the Periodic Schedule Enable bit in the USBCMD register. Software then can poll the Periodic Schedule Status bit to determine when the periodic schedule has made the desired transition. Software must not modify the Periodic Schedule Enable bit unless the value of the Periodic Schedule Enable bit equals that of the Periodic Schedule Status bit.

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 16-49 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.

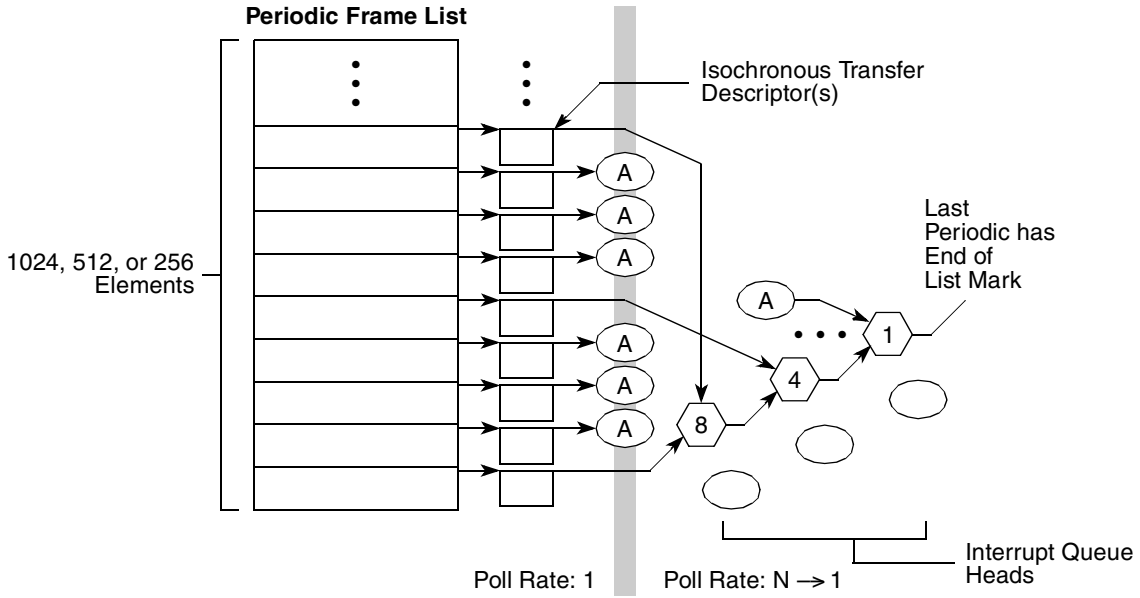


Figure 16-49. Example Periodic Schedule

16.6.8 Managing Isochronous Transfers Using iTDs

The structure of an iTD is presented in Isochronous (High-Speed) Transfer Descriptor (iTD). There are four distinct sections to an iTD:

- The first field is the Next Link Pointer. This field is for schedule linkage purposes only.
- Transaction description array. This area is an eight-element array. Each element represents control and status information for one micro-frame's worth of transactions for a single high-speed isochronous endpoint.
- The buffer page pointer array is a 7-element array of physical memory pointers to data buffers. These are 4K aligned pointers to physical memory.
- Endpoint capabilities. This area utilizes the unused low-order 12 bits of the buffer page pointer array. The fields in this area are used across all transactions executed for this iTD, including endpoint addressing, transfer direction, maximum packet size and high-bandwidth multiplier.

16.6.8.1 Host Controller Operational Model for iTDs

The host controller uses FRINDEX register bits [12–3] to index into the periodic frame list. This means that the host controller visits each frame list element eight consecutive times before incrementing to the next periodic frame list element. Each iTD contains eight transaction descriptions, which map directly to FRINDEX register bits [2–0]. Each iTD can span 8 micro-frames worth of transactions. When the host controller fetches an iTD, it uses FRINDEX register bits [2–0] to index into the transaction description array. If the active bit in the Status field of the indexed transaction description is cleared, the host controller ignores the iTD and follows the Next pointer to the next schedule data structure.

When the indexed active bit is a one the host controller continues to parse the iTD. It stores the indexed transaction description and the general endpoint information (device address, endpoint number, maximum

packet size, etc.). It also uses the Page Select (PG) field to index the buffer pointer array, storing the selected buffer pointer and the next sequential buffer pointer. For example, if PG field is a 0, then the host controller will store Page 0 and Page 1.

The host controller constructs a physical data buffer address by concatenating the current buffer pointer (as selected using the current transaction description's PG field) and the transaction description's Transaction Offset field. The host controller uses the endpoint addressing information and I/O-bit to execute a transaction to the appropriate endpoint. When the transaction is complete, the host controller clears the active bit and writes back any additional status information to the Status field in the currently selected transaction description.

The data buffer associated with the iTD must be virtually contiguous memory. Seven page pointers are provided to support eight high-bandwidth transactions regardless of the starting packet's offset alignment into the first page. A starting buffer pointer (physical memory address) is constructed by concatenating the page pointer (example: page 0 pointer) selected by the active transaction descriptions' PG (example value: 0b00) field with the transaction offset field. As the transaction moves data, the host controller must detect when an increment of the current buffer pointer will cross a page boundary. When this occurs the host controller simply replaces the current buffer pointer's page portion with the next page pointer (example: page 1 pointer) and continues to move data. The size of each bus transaction is determined by the value in the Maximum Packet Size field. An iTD supports high-bandwidth pipes via the Mult (multiplier) field. When the Mult field is 1, 2, or 3, the host controller executes the specified number of Maximum Packet sized bus transactions for the endpoint in the current micro-frame. In other words, the Mult field represents a transaction count for the endpoint in the current micro-frame. If the Mult field is zero, the operation of the host controller is undefined. The transfer description is used to service all transactions indicated by the Mult field.

For OUT transfers, the value of the Transaction *n* Length field represents the total bytes to be sent during the micro-frame. The Mult field must be set by software to be consistent with Transaction *n* Length and Maximum Packet Size. The host controller will send the bytes in Maximum Packet Size'd portions. After each transaction, the host controller decrements it's local copy of Transaction *n* Length by Maximum Packet Size. The number of bytes the host controller sends is always Maximum Packet Size or Transaction *n* Length, whichever is less. The host controller advances the transfer state in the transfer description, updates the appropriate record in the iTD and moves to the next schedule data structure. The maximum sized transaction supported is 3×1024 bytes.

For IN transfers, the host controller issues Mult transactions. It is assumed that software has properly initialized the iTD to accommodate all of the possible data. During each IN transaction, the host controller must use Maximum Packet Size to detect packet babble errors. The host controller keeps the sum of bytes received in the Transaction *n* Length field. After all transactions for the endpoint have completed for the micro-frame, Transaction *n* Length contains the total bytes received. If the final value of Transaction *n* Length is less than the value of Maximum Packet Size, then less data than was allowed for was received from the associated endpoint. This short packet condition does not set the USBINT bit in the USBSTS register. The host controller will not detect this condition. If the device sends more than Transaction *n* Length or Maximum Packet Size bytes (whichever is less), then the host controller will set the Babble Detected bit and clear the Active bit. Note, that the host controller is not required to update the iTD field Transaction *n* Length in this error scenario. If the Mult field is greater than one, then the host controller

will automatically execute the value of Mult transactions. The host controller will not execute all Mult transactions if:

- The endpoint is an OUT and Transaction *n* Length goes to zero before all the Mult transactions have executed (ran out of data), or
- The endpoint is an IN and the endpoint delivers a short packet, or an error occurs on a transaction before Mult transactions have been executed. The end of micro-frame may occur before all of the transaction opportunities have been executed. When this happens, the transfer state of the transfer description is advanced to reflect the progress that was made, the result written back to the iTD and the host controller proceeds to processing the next micro-frame.

16.6.8.2 Software Operational Model for iTDs

A client buffer request to an isochronous endpoint may span 1 to N micro-frames. When N is larger than one, system software may have to use multiple iTDs to read or write data with the buffer (if N is larger than eight, it must use more than one iTD).

Figure 16-50 illustrates the simple model of how a client buffer is mapped by system software to the periodic schedule (that is, the periodic frame list and a set of iTDs). On the right is the client description of its request. The description includes a buffer base address plus additional annotations to identify which portions of the buffer should be used with each bus transaction. In the middle is the iTD data structures used by the system software to service the client request. Each iTD can be initialized to service up to 24 transactions, organized into eight groups of up to three transactions each. Each group maps to one micro-frame's worth of transactions. The EHCI controller does not provide per-transaction results within a micro-frame. It treats the per-micro-frame transactions as a single logical transfer. On the left is the host controller’s frame list. System software establishes references from the appropriate locations in the frame list to each of the appropriate iTDs. If the buffer is large, then system software can use a small set of iTDs to service the entire buffer. System software can activate the transaction description records (contained in each iTD) in any pattern required for the particular data stream.

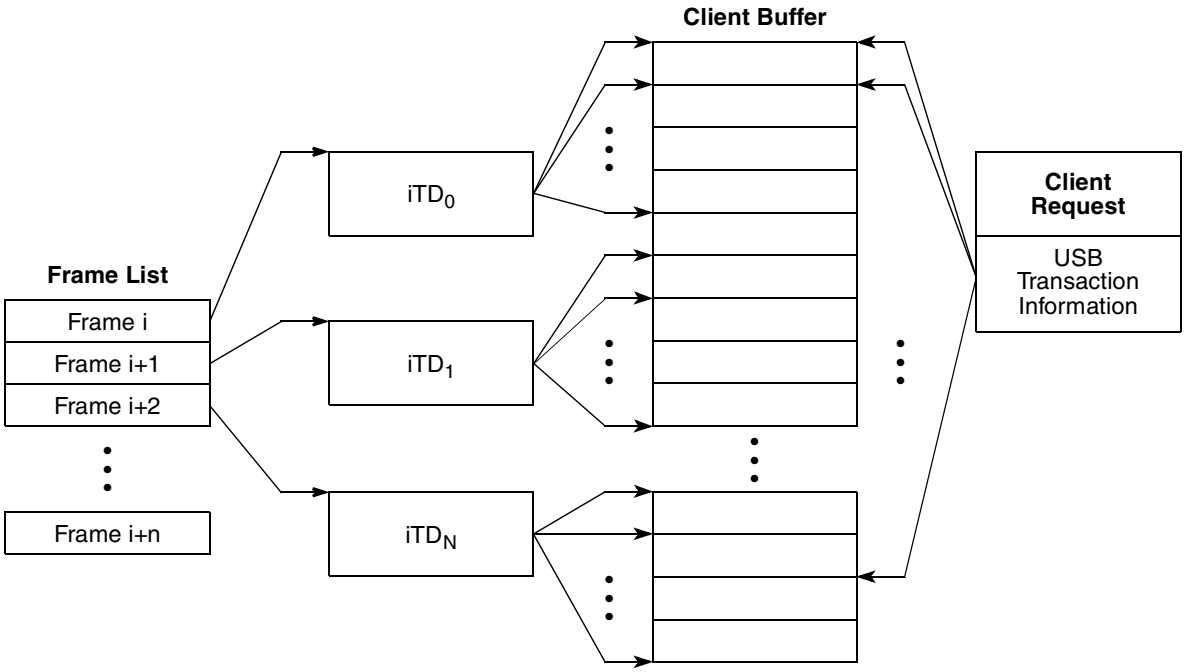


Figure 16-50. Example Association of iTDs to Client Request Buffer

As noted above, the client request includes a pointer to the base of the buffer and offsets into the buffer to annotate which buffer sections are to be used on each bus transaction that occurs on this endpoint. System software must initialize each transaction description in an iTD to ensure it uses the correct portion of the client buffer. For example, for each transaction description, the PG field is set to index the correct physical buffer page pointer and the Transaction Offset field is set relative to the correct buffer pointer page (for example, the same one referenced by the PG field). When the host controller executes a transaction it selects a transaction description record based on FRINDEX[2–0]. It then uses the current Page Buffer Pointer (as selected by the PG field) and concatenates to the transaction offset field. The result is a starting buffer address for the transaction. As the host controller moves data for the transaction, it must watch for a page wrap condition and properly advance to the next available Page Buffer Pointer. System software must not use the Page 6 buffer pointer in a transaction description where the length of the transfer will wrap a page boundary. Doing so yields undefined behavior. The host controller hardware is not required to alias the page selector to page zero. USB 2.0 isochronous endpoints can specify a period greater than one. Software can achieve the appropriate scheduling by linking iTDs into the appropriate frames (relative to the frame list) and by setting appropriate transaction description elements active bits to a one.

16.6.8.2.1 Periodic Scheduling Threshold

The Isochronous Scheduling Threshold field in the HCCPARAMS capability register is an indicator to system software as to how the host controller pre-fetches and effectively caches schedule data structures. It is used by system software when adding isochronous work items to the periodic schedule. The value of this field indicates to system software the minimum distance it can update isochronous data (relative to the current location of the host controller execution in the periodic list) and still have the host controller process them.

The iTD and siTD data structures each describe 8 micro-frames worth of transactions. The host controller is allowed to cache one (or more) of these data structures in order to reduce memory traffic. There are three basic caching models that account for the fact the isochronous data structures span 8 micro-frames. The three caching models are: no caching, micro-frame caching and frame caching.

When software is adding new isochronous transactions to the schedule, it always performs a read of the FRINDEX register to determine the current frame and micro-frame the host controller is currently executing. Of course, there is no information about where in the micro-frame the host controller is, so a constant uncertainty factor of one micro-frame has to be assumed. Combining the knowledge of where the host controller is executing with the knowledge of the caching model allows the definition of simple algorithms for how closely software can reliably work to the executing host controller.

No caching is indicated with a value of zero in the Isochronous Scheduling Threshold field. The host controller may pre-fetch data structures during a periodic schedule traversal (per micro-frame) but will always dump any accumulated schedule state at the end of the micro-frame. At the appropriate time relative to the beginning of every micro-frame, the host controller always begins schedule traversal from the frame list. Software can use the value of the FRINDEX register (plus the constant 1 uncertainty-factor) to determine the approximate position of the executing host controller. When no caching is selected, software can add an isochronous transaction as near as 2 micro-frames in front of the current executing position of the host controller.

Frame caching is indicated with a non-zero value in bit [7] of the Isochronous Scheduling Threshold field. In the frame-caching model, system software assumes that the host controller caches one (or more) isochronous data structures for an entire frame (8 micro-frames). Software uses the value of the FRINDEX register (plus the constant 1 uncertainty) to determine the current micro-frame/frame (assume modulo 8 arithmetic in adding the constant 1 to the micro-frame number). For any current frame N, if the current micro-frame is 0 to 6, then software can safely add isochronous transactions to Frame N + 1. If the current micro-frame is 7, then software can add isochronous transactions to Frame N + 2.

Micro-frame caching is indicated with a non-zero value in the least-significant 3 bits of the Isochronous Scheduling Threshold field. System software assumes the host controller caches one or more periodic data structures for the number of micro-frames indicated in the Isochronous Scheduling Threshold field. For example, if the count value were 2, then the host controller keeps a window of 2 micro-frames worth of state (current micro-frame, plus the next) on-chip. On each micro-frame boundary, the host controller releases the current micro-frame state and begins accumulating the next micro-frame state.

16.6.9 Asynchronous Schedule

The Asynchronous schedule traversal is enabled or disabled via the Asynchronous Schedule Enable bit in the USBCMD register. If the Asynchronous Schedule Enable bit is cleared, then the host controller simply does not try to access the asynchronous schedule via the ASYNCLISTADDR register. Likewise, if the Asynchronous Schedule Enable bit is set, the host controller does use the ASYNCLISTADDR register to traverse the asynchronous schedule. Modifications to the Asynchronous Schedule Enable bit 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.

The Asynchronous Schedule Status bit in the USBSTS register indicates status of the asynchronous schedule. System software enables (or disables) the asynchronous schedule by writing a one (or zero) to

the Asynchronous Schedule Enable bit in the USBCMD register. Software then can poll the Asynchronous Schedule Status bit to determine when the asynchronous schedule has made the desired transition. Software must not modify the Asynchronous Schedule Enable bit unless the value of the Asynchronous Schedule Enable bit equals that of the Asynchronous Schedule Status bit.

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 the Asynchronous Schedule Enable bit is cleared.

Software may only write this register with defined results when the schedule is disabled, for example, Asynchronous Schedule Enable bit in the USBCMD and the Asynchronous Schedule Status bit in the USBSTS register 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 the Asynchronous Schedule Enable bit is set. The asynchronous schedule is actually enabled when the Asynchronous Schedule Status bit is set.

When the host controller begins servicing the asynchronous schedule, it begins by using the value of the ASYNCLISTADDR register. It reads the first referenced data structure and begins executing transactions and traversing the linked list as appropriate. When the host controller completes processing the asynchronous schedule, it retains the value of the last accessed queue head's horizontal pointer in the ASYNCLISTADDR register. Next time the asynchronous schedule is accessed, this is the first data structure that is serviced. This provides round-robin fairness for processing the asynchronous schedule.

A host controller completes processing the asynchronous schedule when one of the following events occur:

- The end of a micro-frame occurs.
- The host controller detects an empty list condition
- The schedule has been disabled via the Asynchronous Schedule Enable bit in the USBCMD register.

The queue heads in the asynchronous list are linked into a simple circular list as shown in Figure 16-46. Queue head data structures are the only valid data structures that may be linked into the asynchronous schedule. An isochronous transfer descriptor (iTD or siTD) in the asynchronous schedule yields undefined results.

The maximum packet size field in a queue head is sized to accommodate the use of this data structure for all non-isochronous transfer types. The USB Specification, Revision 2.0 specifies the maximum packet sizes for all transfer types and transfer speeds. System software should always parameterize the queue head data structures according to the core specification requirements.

16.6.9.1 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 the Asynchronous Schedule Enable bit in the USBCMD register 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
```

16.6.9.2 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 the Asynchronous Schedule Enable bit in the USBCMD register to a zero. Software can determine when the list is idle when the Asynchronous Schedule Status bit in the USBSTS register 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 (Interrupt on Async Advance Doorbell bit in the USBCMD register) that allows software to inform the host controller that something has been removed from its asynchronous schedule. The second bit is a status bit (Interrupt on Async Advance bit in the USBSTS register) 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 (Interrupt on Async Advance bit in the USBINTR register) 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 16-51 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).

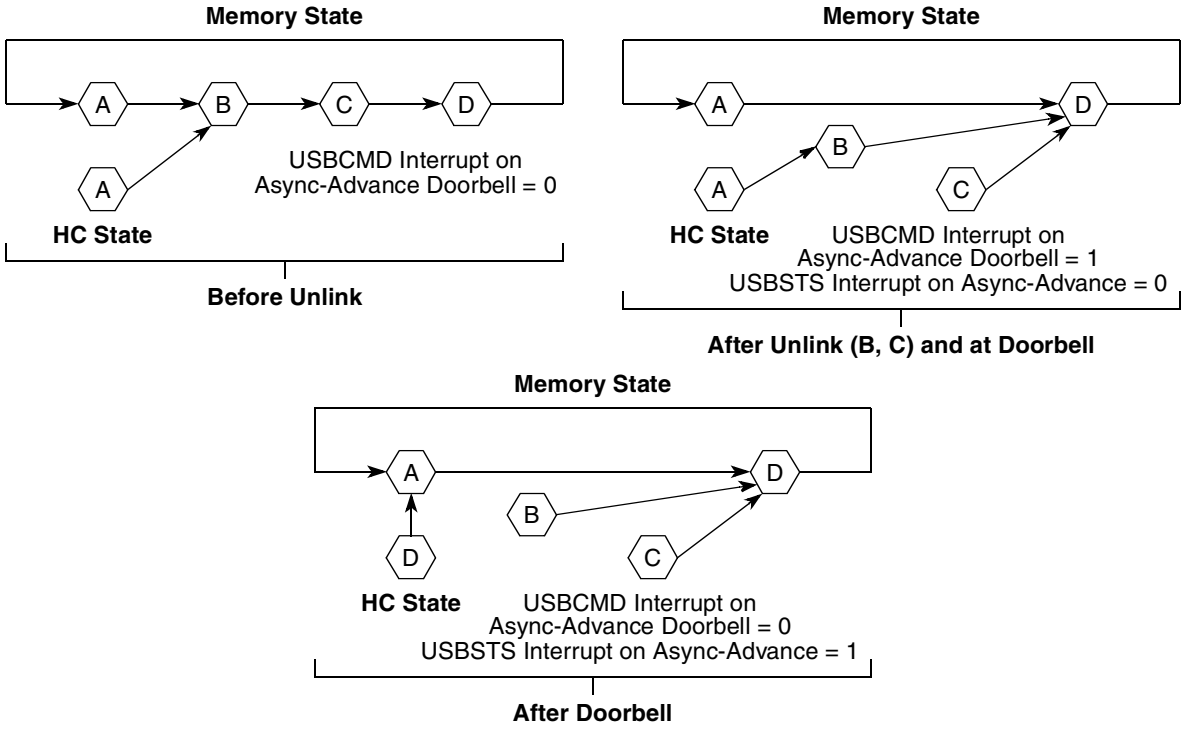


Figure 16-51. Generic Queue Head Unlink Scenario

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 the Advance on Async status bit.

Software may re-use the memory associated with the removed queue heads after it observes the Interrupt on Async Advance status bit 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

16.6.9.3 Empty Asynchronous Schedule Detection

EHCI uses two bits to detect when the asynchronous schedule is empty. The queue head data structure (see Figure 16-43) 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 16.6.9.4, “Asynchronous Schedule Traversal: Start Event.”

If the controller ever encounters an H-bit of one and a Reclamation bit of zero, the controller simply stops traversal of the asynchronous schedule.

An example illustrating the H-bit in a schedule is shown in Figure 16-52

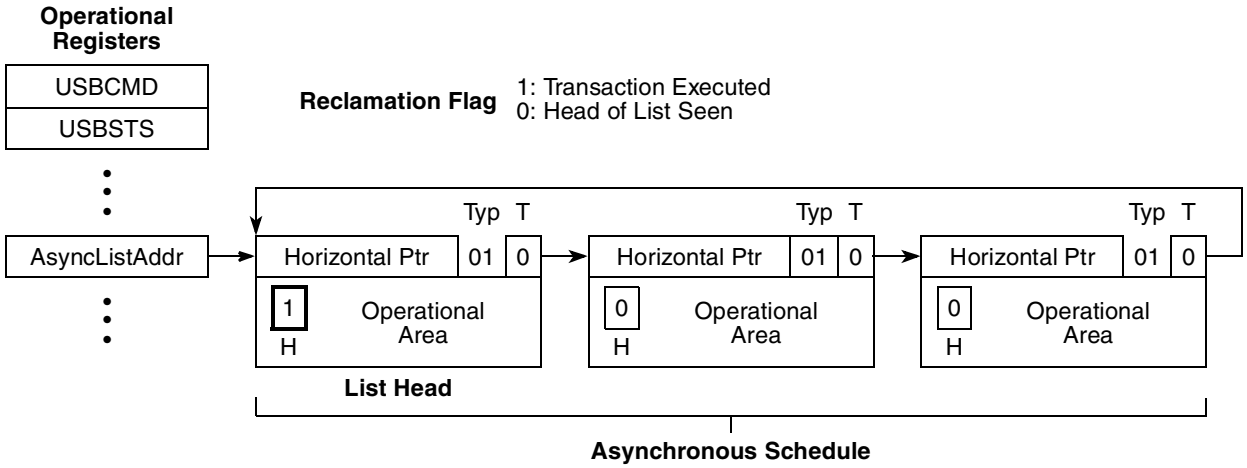


Figure 16-52. Asynchronous Schedule List with Annotation to Mark Head of List

16.6.9.4 Asynchronous Schedule Traversal: Start Event

Once the host controller has idled itself using the empty schedule detection, it naturally activates and begins processing from the Periodic Schedule at the beginning of each micro-frame. In addition, it may have idled itself early in a micro-frame. When this occurs (idles early in the micro-frame) the host controller must occasionally reactivate during the micro-frame and traverse the asynchronous schedule to determine whether any progress can be made. Asynchronous schedule Start Events are defined to be:

- Whenever the host controller transitions from the periodic schedule to the asynchronous schedule. If the periodic schedule is disabled and the asynchronous schedule is enabled, then the beginning of the micro-frame is equivalent to the transition from the periodic schedule, or
- The asynchronous schedule traversal restarts from a sleeping state.

16.6.9.5 Reclamation Status Bit (USBSTS Register)

The operation of the empty asynchronous schedule detection feature depends on the proper management of the Reclamation bit 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 the Reclamation bit whenever an asynchronous schedule traversal Start Event occurs. The Reclamation bit is also set whenever the host controller executes a transaction while traversing the asynchronous schedule.The host controller clears the Reclamation bit 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 the Reclamation bit when executing from the periodic schedule.

16.6.10 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 16.5.5, “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.

16.6.10.1 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 16-53 illustrates these requirements.

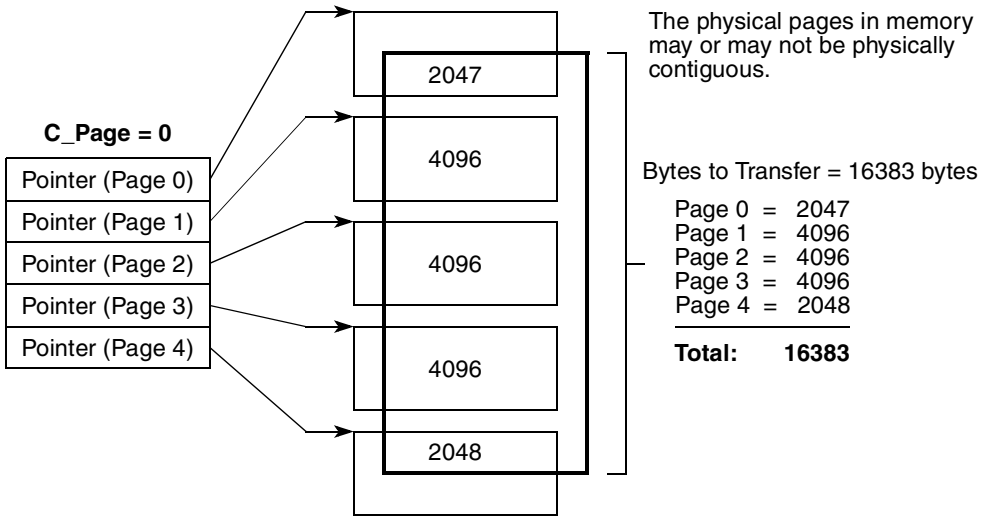


Figure 16-53. Example Mapping of qTD Buffer Pointers to Buffer Pages

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 16-53 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.

16.6.10.2 Adding Interrupt Queue Heads to the Periodic Schedule

The link path(s) from the periodic frame list to a queue head establishes in which frames a transaction can be executed for the queue head. Queue heads are linked into the periodic schedule so they are polled at the appropriate rate. System software sets a bit in a queue head's S-Mask to indicate which micro-frame within a 1 millisecond period a transaction should be executed for the queue head. Software must ensure that all queue heads in the periodic schedule have S-Mask set to a non-zero value. An S-mask with a zero value in the context of the periodic schedule yields undefined results.

If the desired poll rate is greater than one frame, system software can use a combination of queue head linking and S-Mask values to spread interrupts of equal poll rates through the schedule so that the periodic bandwidth is allocated and managed in the most efficient manner possible. Some examples are illustrated in Table 16-72.

Table 16-72. Example Periodic Reference Patterns for Interrupt Transfers

Frame # Reference Sequence	Description
0, 2, 4, 6, 8, ... S-Mask = 0x01	A queue head for the bInterval of 2 milliseconds (16 micro-frames) is linked into the periodic schedule so that it is reachable from the periodic frame list locations indicated in the previous column. In addition, the S-Mask field in the queue head is set to 0x01, indicating that the transaction for the endpoint should be executed on the bus during micro-frame 0 of the frame.
0, 2, 4, 6, 8, ... S-Mask = 0x02	Another example of a queue head with a bInterval of 2 milliseconds is linked into the periodic frame list at exactly the same interval as the previous example. However, the S-Mask is set to 0x02 indicating that the transaction for the endpoint should be executed on the bus during micro-frame 1 of the frame.

16.6.10.3 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.

16.6.11 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 16-60). 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 16-73 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 16-73. Ping Control State Transition Table

	Event		
Current	**Host**	**Device**	**Next**
Do Ping	PING	Nak	Do Ping
Do Ping	PING	Ack	Do OUT
Do Ping	PING	XactErr[1]	Do Ping
Do Ping	PING	Stall	N/C[2]
Do OUT	OUT	Nak	Do Ping
Do OUT	OUT	Nyet	Do Ping[3]
Do OUT	OUT	Ack	Do OUT
Do OUT	OUT	XactErr[1]	Do Ping
Do OUT	OUT	Stall	N/C[2]

1 Transaction Error (XactErr) is any time the host misses the handshake.

2 No transition change required for the Ping State bit. The Stall handshake results in the endpoint being halted (for example, Active cleared and Halt set). Software intervention is required to restart queue.

3 A Nyet response to an OUT means that the device has accepted the data, but cannot receive any more at this time. Host must advance the transfer state and additionally, transition the Ping State bit to Do Ping.

The Ping State bit is described in Table 16-60. The defined ping protocol allows the host to be imprecise on the initialization of the ping protocol (that is, start in Do OUT when we don't know whether there is space on the device or not). The host controller manages the Ping State bit. System software sets the initial value in the queue head when it initializes a queue head. The host controller preserves the Ping State bit

across all queue advancements. This means that when a new qTD is written into the queue head overlay area, the previous value of the Ping State bit is preserved.

16.6.12 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.

16.6.12.1 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.

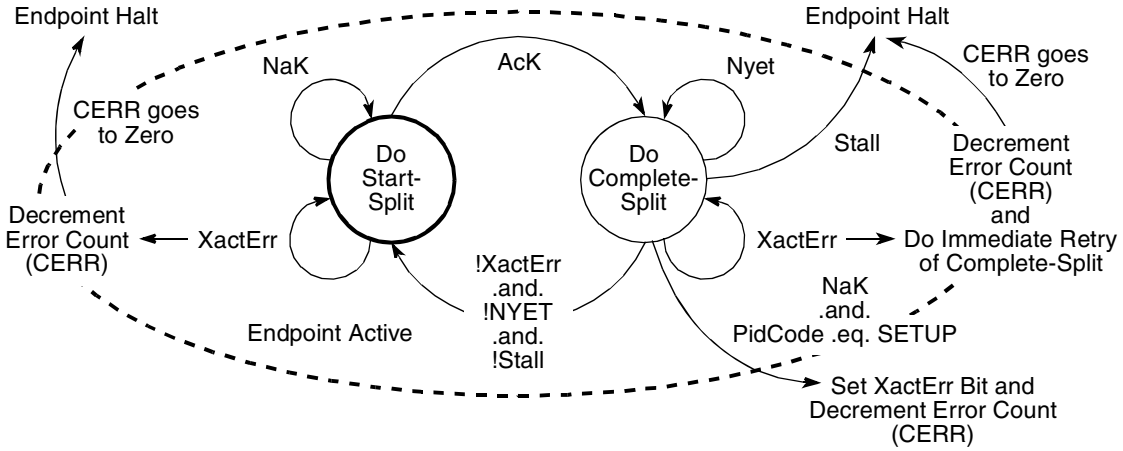


Figure 16-54. Host Controller Asynchronous Schedule Split-Transaction State Machine

16.6.12.1.1 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.

16.6.12.1.2 Asynchronous - Do-Complete-Split

This state is entered from the Do-Start-Split state only after a start-split transaction receives an Ack handshake from the transaction translator.

For queue heads in this state, the host controller executes a complete-split transaction to the transaction translator. If the transaction translator responds with a Nyet handshake, the queue head is left in this state, the error counter is reset and the host controller proceeds to the next queue head in the asynchronous schedule. When a Nyet handshake is received for a bus transaction where the queue head's PID Code indicates an IN or OUT, the host controller reloads the error counter (Cerr). When a Nyet handshake is received for a complete-split bus transaction where the queue head's PID Code indicates a SETUP, the host controller must not adjust the value of Cerr.

Independent of PID Code, the following responses have the indicated effects:

- Transaction Error (XactErr). Timeout/data CRC failure. The error counter (Cerr) is decremented by one and the complete split transaction is immediately retried (if possible). If there is not enough time in the micro-frame to execute the retry, the host controller ensures that the next time the host

controller begins executing from the Asynchronous schedule, it must begin executing from this queue head. If another start-split (for some other endpoint) is sent to the transaction translator before the complete-split is really completed, the transaction translator could dump the results (which were never delivered to the host). This is why the core specification states the retries must be immediate. When the host controller returns to the asynchronous schedule in the next micro-frame, the first transaction from the schedule will be the retry for this endpoint. If Cerr went to zero, the host controller halts the queue.

- NAK. The target endpoint Nak'd the full- or low-speed transaction. The state of the transfer is not advanced and the state is exited. If the PID Code is a SETUP, then the Nak response is a protocol error. The XactErr status bit is set and the Cerr field is decremented.
- STALL. The target endpoint responded with a STALL handshake. The host controller sets the halt bit in the status byte, retires the qTD but does not attempt to advance the queue.

If the PID Code indicates an IN, then any of following responses are expected:

- DATA0/1. On reception of data, the host controller ensures the PID matches the expected data toggle and checks CRC. If the packet is good, the host controller advances the state of the transfer (for example, moves the data pointer by the number of bytes received, decrements the BytesToTransfer field by the number of bytes received, and toggles the dt bit). The host controller then exits this state. The response and advancement of transfer may trigger other processing events, such as retirement of the qTD and advancement of the queue.

  If the data sequence PID does not match the expected, the data is ignored, the transfer state is not advanced and this state is exited.

If the PID Code indicates an OUT/SETUP, then any of following responses are expected:

- ACK. The target endpoint accepted the data, so the host controller must advance the state of the transfer. The Current Offset field is incremented by Maximum Packet Length or Bytes to Transfer, whichever is less. The Bytes To Transfer field is decremented by the same amount and the data toggle bit (dt) is toggled. The host controller then exits this state.

  Advancing the transfer state may cause other processing events such as retirement of the qTD and advancement of the queue.

16.6.12.2 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.

16.6.12.2.1 Split Transaction Scheduling Mechanisms for Interrupt

Full- and low-speed Interrupt queue heads have an EPS field indicating full- or low-speed and have a non-zero S-mask field. The host controller can detect this combination of parameters and assume the endpoint is a periodic endpoint. Low- and full-speed interrupt queue heads require the use of the split transaction protocol. The host controller sets the Endpoint Type (ET) field in the split token to indicate the transaction is an interrupt. These transactions are managed through a transaction translator's periodic pipeline. Software should not set these fields to indicate the queue head is an interrupt unless the queue head is used in the periodic schedule.

System software manages the per/transaction translator periodic pipeline by budgeting and scheduling exactly during which micro-frames the start-splits and complete-splits for each endpoint will occur. The characteristics of the transaction translator are such that the high-speed transaction protocol must execute during explicit micro-frames, or the data or response information in the pipeline is lost. Figure 16-55 illustrates the general scheduling boundary conditions that are supported by the EHCI periodic schedule and queue head data structure. The S and C_n labels indicate micro-frames where software can schedule start-splits and complete splits (respectively).

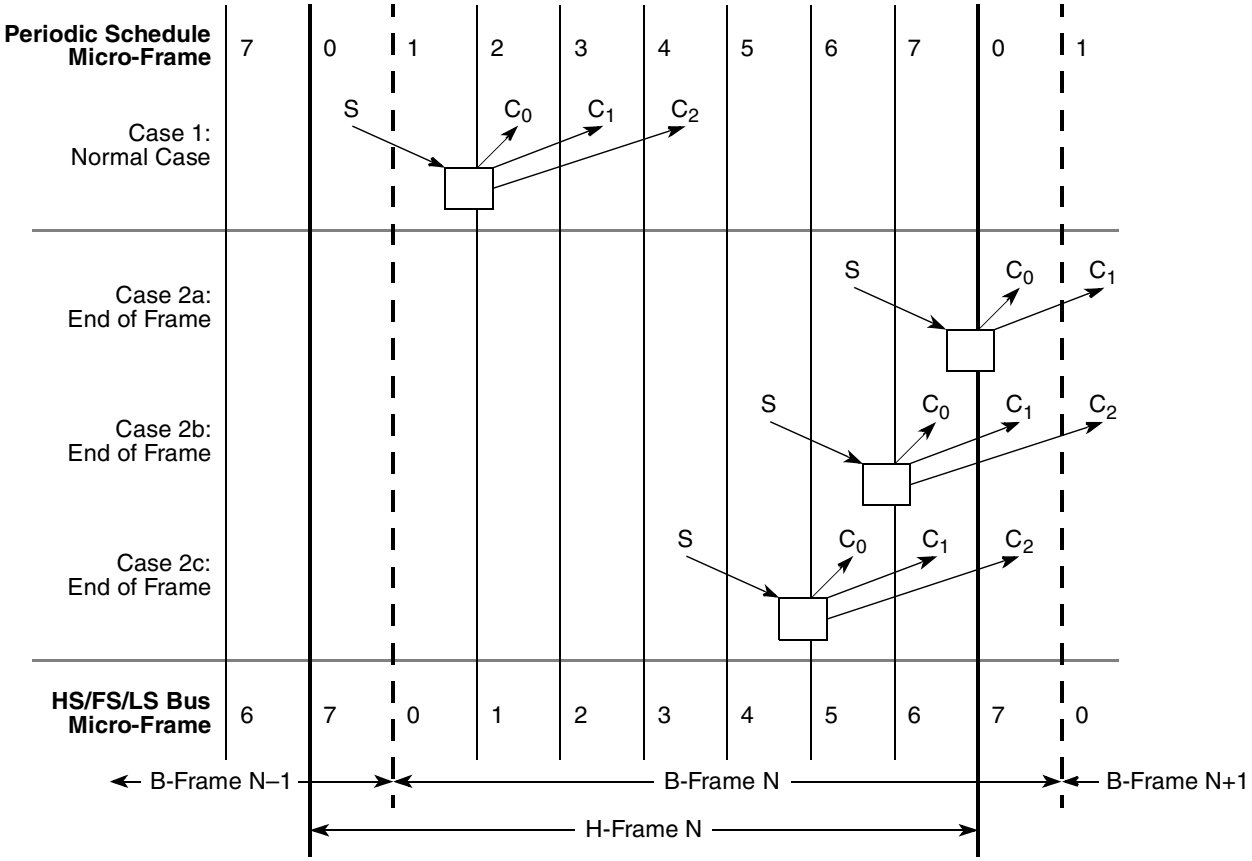


Figure 16-55. Split Transaction, Interrupt Scheduling Boundary Conditions

The scheduling cases are:

- Case 1: The normal scheduling case is where the entire split transaction is completely bounded by a frame (H-Frame in this case).
- Case 2a through Case 2c: The USB 2.0 hub pipeline rules states clearly, when and how many complete-splits must be scheduled to account for earliest to latest execution on the full/low-speed link. The complete-splits may span the H-Frame boundary when the start-split is in micro-frame 4 or later. When this occurs, the H-Frame to B-Frame alignment requires that the queue head be reachable from consecutive periodic frame list locations. System software cannot build an efficient schedule that satisfies this requirement unless it uses FSTNs. Figure 16-56 illustrates the general layout of the periodic schedule.

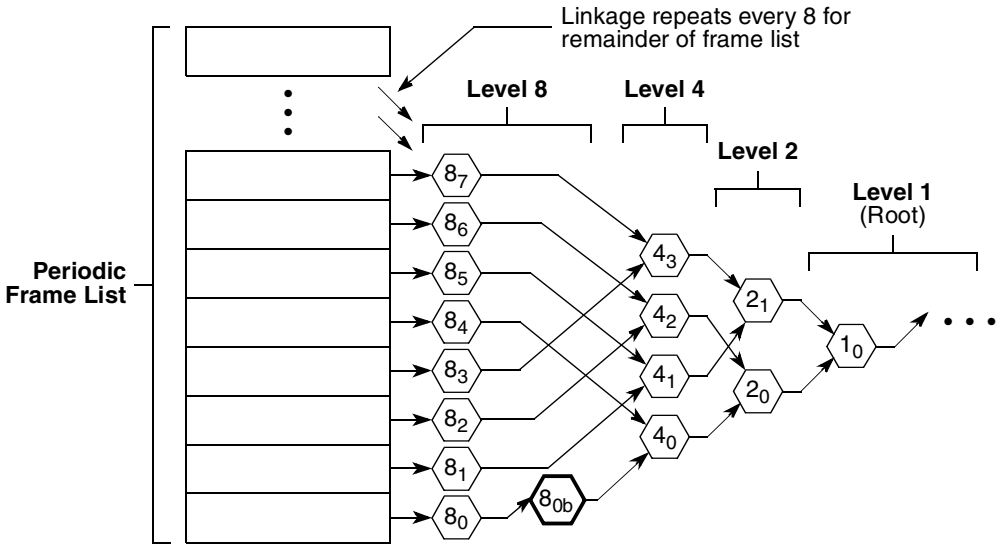


Figure 16-56. General Structure of EHCI Periodic Schedule Utilizing Interrupt Spreading

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 16.5.7, “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 16-60). This bit is used to track the current state of the split transaction.
- Frame S-mask. This is a bit-field where-in system software sets a bit corresponding to the micro-frame (within an H-Frame) that the host controller should execute a start-split transaction. This is always qualified by the value of the SplitXState bit in the Status field of the queue head. For example, referring to Figure 16-55, case one, the S-mask would have a value of 0b0000_0001 indicating that if the queue head is traversed by the host controller, and the SplitXState indicates Do_Start, and the current micro-frame as indicated by FRINDEX[2–0] is 0, then execute a start-split transaction.
- Frame C-mask. This is a bit-field where system software sets one or more bits corresponding to the micro-frames (within an H-Frame) that the host controller should execute complete-split transactions. The interpretation of this field is always qualified by the value of the SplitXState bit in the Status field of the queue head. For example, referring to Figure 16-55, case one, the C-mask would have a value of 0b0001_1100 indicating that if the queue head is traversed by the host controller, and the SplitXState indicates Do_Complete, and the current micro-frame as indicated by FRINDEX[2–0] is 2, 3, or 4, then execute a complete-split transaction. It is software's responsibility to ensure that the translation between H-Frames and B-Frames is correctly performed when setting bits in S-mask and C-mask.

16.6.12.2.2 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 16.5.7, “Periodic Frame Span Traversal Node (FSTN).”
- A Save Place indicator; this is always an FSTN with its Back Path Link Pointer[T] bit cleared.
- A Restore indicator; this is always an FSTN with its Back Path Link Pointer[T] bit set.
- Host controller FSTN traversal rules.

When the host controller encounters an FSTN during micro-frames 2 through 7 it simply follows the node's Normal Path Link Pointer to access the next schedule data structure. Note that the FSTN's Normal Path Link Pointer[T] bit may set, which the host controller must interpret as the end of periodic list mark.

When the host controller encounters a Save-Place FSTN in micro-frames 0 or 1, it saves the value of the Normal Path Link Pointer and sets an internal flag indicating that it is executing in Recovery Path mode. Recovery Path mode modifies the host controller's rules for how it traverses the schedule and limits which data structures are considered for execution of bus transactions. The host controller continues executing in

Recovery Path mode until it encounters a Restore FSTN or it determines that it has reached the end of the micro-frame.

The rules for schedule traversal and limited execution while in Recovery Path mode are:

- Always follow the Normal Path Link Pointer when it encounters an FSTN that is a Save-Place indicator. The host controller must not recursively follow Save-Place FSTNs. Therefore, while executing in Recovery Path mode, it must never follow an FSTN's Back Path Link Pointer.
- Do not process an siTD or iTD data structure; simply follow its Next Link Pointer.
- Do not process a QH (Queue Head) whose EPS field indicates a high-speed device; simply follow its Horizontal Link Pointer.
- When a QH's EPS field indicates a Full/Low-speed device, the host controller only considers it for execution if its SplitXState is DoComplete (note: this applies whether the PID Code indicates an IN or an OUT). Refer to the *EHCI Specification* for a complete list of additional conditions that must be met in general for the host controller to issue a bus transaction. Note that the host controller must not execute a Start-split transaction while executing in Recovery Path mode. Refer to the *EHCI Specification* for special handling when in Recovery Path mode.
- Stop traversing the recovery path when it encounters an FSTN that is a Restore indicator. The host controller unconditionally uses the saved value of the Save-Place FSTN's Normal Path Link Pointer when returning to the normal path traversal. The host controller must clear the context of executing a Recovery Path when it restores schedule traversal to the Save-Place FSTN's Normal Path Link Pointer.

  If the host controller determines that there is not enough time left in the micro-frame to complete processing of the periodic schedule, it abandons traversal of the recovery path, and clears the context of executing a recovery path. The result is that at the start of the next consecutive micro-frame, the host controller starts traversal at the frame list.

An example traversal of a periodic schedule that includes FSTNs is illustrated in Figure 16-57.

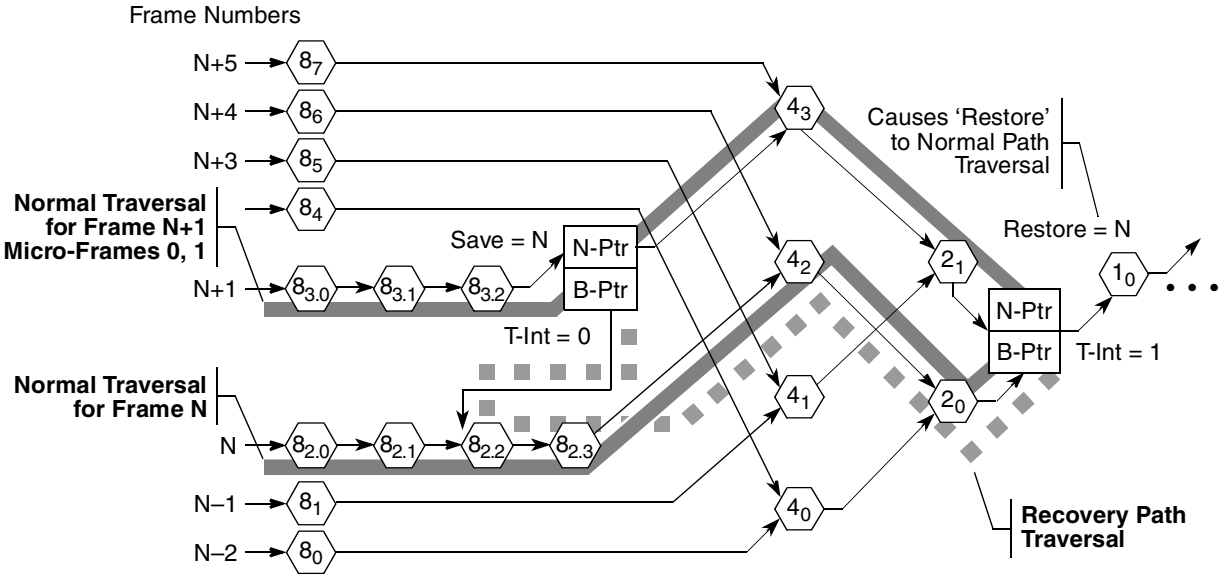


Figure 16-57. Example Host Controller Traversal of Recovery Path via FSTNs

In frame N (micro-frames 0-7), for this example, the host controller traverses all of the schedule data structures utilizing the Normal Path Link Pointers in any FSTNs it encounters. This is because the host controller has not yet encountered a Save-Place FSTN so it is not executing in Recovery Path mode. When it encounters the Restore FSTN, (Restore-N), during micro-frames 0 and 1, it uses Restore-N. Normal Path Link Pointer to traverse to the next data structure (that is, normal schedule traversal). This is because the host controller must use a Restore FSTN's Normal Path Link Pointer when not executing in a Recovery-Path mode. The nodes traversed during frame N include: {$8_{2.0}$, $8_{2.1}$, $8_{2.2}$, $8_{2.3}$, 4_2, 2_0, Restore-N, 1_0 ...}.

In frame N+1 (micro-frames 0 and 1), when the host controller encounters Save-Path FSTN (Save-N), it observes that Save-N.Back Path Link Pointer.T-bit is zero (definition of a Save-Path indicator). The host controller saves the value of Save-N. Normal Path Link Pointer and follows Save-N.Back Path Link Pointer. At the same time, it sets an internal flag indicating that it is now in Recovery Path mode (the recovery path is annotated in Figure 16-57 with a large dashed line). The host controller continues traversing data structures on the recovery path and executing only those bus transactions as noted above, on the recovery path until it reaches Restore FSTN (Restore-N). Restore-N.Back Path Link Pointer.T-bit is set (definition of a Restore indicator), so the host controller exits Recovery Path mode by clearing the internal Recovery Path mode flag and commences (restores) schedule traversal using the saved value of the Save-Place FSTN's Normal Path Link Pointer (for example, Save-N.Normal Path Link Pointer). The nodes traversed during these micro-frames include: {$8_{3.0}$, $8_{3.1}$, $8_{3.2}$, Save-A, $8_{2.2}$, $8_{2.3}$, 4_2, 2_0, Restore-N, 4_3, 2_1, Restore-N, 10 ...}.

In frame N+1 (micro-frames 2-7), when the host controller encounters Save-Path FSTN Save-N, it unconditionally follows Save-N.Normal Path Link Pointer. The nodes traversed during these micro-frames include: {$8_{3.0}$, $8_{3.1}$, $8_{3.2}$, Save-A, 4_3, 2_1, Restore-N, 1_0 ...}.

16.6.12.2.3 Software Operational Model for FSTNs

Software must create a consistent, coherent schedule for the host controller to traverse. When using FSTNs, system software must adhere to the following rules:

- Each Save-Place indicator requires a matching Restore indicator.

  The Save-Place indicator is an FSTN with a valid Back Path Link Pointer and T-bit equal to zero. Note that Back Path Link Pointer[Typ] field must be set to indicate the referenced data structure is a queue head. The Restore indicator is an FSTN with its Back Path Link Pointer[T] bit set.

  A Restore FSTN may be matched to one or more Save-Place FSTNs. For example, if the schedule includes a poll-rate 1 level, then system software only needs to place a Restore FSTN at the beginning of this list in order to match all possible Save-Place FSTNs.
- If the schedule does not have elements linked at a poll-rate level of one, and one or more Save-Place FSTNs are used, then System Software must ensure the Restore FSTN's Normal Path Link Pointer's T-bit is set, as this will be use to mark the end of the periodic list.
- When the schedule does have elements linked at a poll rate level of one, a Restore FSTN must be the first data structure on the poll rate one list. All traversal paths from the frame list converge on the poll-rate one list. System software must ensure that Recovery Path mode is exited before the host controller is allowed to traverse the poll rate level one list.

- A Save-Place FSTN's Back Path Link Pointer must reference a queue head data structure. The referenced queue head must be reachable from the previous frame list location. In other words, if the Save-Place FSTN is reachable from frame list offset N, then the FSTN's Back Path Link Pointer must reference a queue head that is reachable from frame list offset N-1.

Software should make the schedule as efficient as possible. What this means in this context is that software should have no more than one Save-Place FSTN reachable in any single frame. Note there will be times when two (or more, depending on the implementation) could exist as full/low-speed footprints change with bandwidth adjustments. This could occur, for example when a bandwidth rebalance causes system software to move the Save-Place FSTN from one poll rate level to another. During the transition, software must preserve the integrity of the previous schedule until the new schedule is in place.

16.6.12.2.4 Tracking Split Transaction Progress for Interrupt Transfers

To correctly maintain the data stream, the host controller must be able to detect and report errors where data is lost. For interrupt-IN transfers, data is lost when it makes it into the USB 2.0 hub, but the USB 2.0 host system is unable to get it from the USB 2.0 hub and into the system before it expires from the transaction translator pipeline. When a lost data condition is detected, the queue is halted, thus signaling system software to recover from the error. A data-loss condition exists whenever a start-split is issued, accepted and successfully executed by the USB 2.0 hub, but the complete-splits get unrecoverable errors on the high-speed link, or the complete-splits do not occur at the correct times. One reason complete-splits might not occur at the right time would be due to host-induced system hold-offs that cause the host controller to miss bus transactions because it cannot get timely access to the schedule in system memory.

The same condition can occur for an interrupt-OUT, but the result is not an endpoint halt condition, but rather effects only the progress of the transfer. The queue head has the following fields to track the progress of each split transaction. These fields are used to keep incremental state about which (and when) portions have been executed.

- C-prog-mask. This is an eight-bit bit-vector where the host controller keeps track of which complete-splits have been executed. Due to the nature of the Transaction Translator periodic pipeline, the complete-splits need to be executed in-order. The host controller needs to detect when the complete-splits have not been executed in order. This can only occur due to system hold-offs where the host controller cannot get to the memory-based schedule. C-prog-mask is a simple bit-vector that the host controller sets one of the C-prog-mask bits for each complete-split executed. The bit position is determined by the micro-frame number in which the complete-split was executed. The host controller always checks C-prog-mask before executing a complete-split transaction. If the previous complete-splits have not been executed then it means one (or more) have been skipped and data has potentially been lost.
- FrameTag. This field is used by the host controller during the complete-split portion of the split transaction to tag the queue head with the frame number (H-Frame number) when the next complete split must be executed.
- S-bytes. This field can be used to store the number of data payload bytes sent during the start-split (if the transaction was an OUT). The S-bytes field must be used to accumulate the data payload bytes received during the complete-splits (for an IN).

16.6.12.2.5 Split Transaction Execution State Machine for Interrupt

In the following section, all references to micro-frame are in the context of a micro-frame within an H-Frame.

As with asynchronous Full- and Low-speed endpoints, a split-transaction state machine is used to manage the split transaction sequence. Aside from the fields defined in the queue head for scheduling and tracking the split transaction, the host controller calculates one internal mechanism that is also used to manage the split transaction. The internal calculated mechanism is:

- cMicroFrameBit. This is a single-bit encoding of the current micro-frame number. It is an eight-bit value calculated by the host controller at the beginning of every micro-frame. It is calculated from the three least significant bits of the FRINDEX register (that is, cMicroFrameBit = (1 shifted-left(FRINDEX[2–0]))). The cMicroFrameBit has at most one bit asserted, which always corresponds to the current micro-frame number. For example, if the current micro-frame is 0, then cMicroFrameBit will equal 0b0000_0001.

  The variable cMicroFrameBit is used to compare against the S-mask and C-mask fields to determine whether the queue head is marked for a start- or complete-split transaction for the current micro-frame.

Figure 16-58 illustrates how a complete interrupt split transaction is managed. There are two phases to each split transaction. The first is a single start-split transaction, which occurs when the SplitXState is at Do_Start and the single bit in cMicroFrameBit has a corresponding bit active in QH[S-mask]. The transaction translator does not acknowledge the receipt of the periodic start-split, so the host controller unconditionally transitions the state to Do_Complete. Due to the available jitter in the transaction translator pipeline, there will be more than one complete-split transaction scheduled by software for the Do_Complete state. This translates simply to the fact that there are multiple bits set in the QH[C-mask] field.

The host controller keeps the queue head in the Do_Complete state until the split transaction is complete (see definition below), or an error condition triggers the three-strikes-rule (for example, after the host tries the same transaction three times, and each encounters an error, the host controller stops retrying the bus transaction and halts the endpoint, thus requiring system software to detect the condition and perform system-dependent recovery).

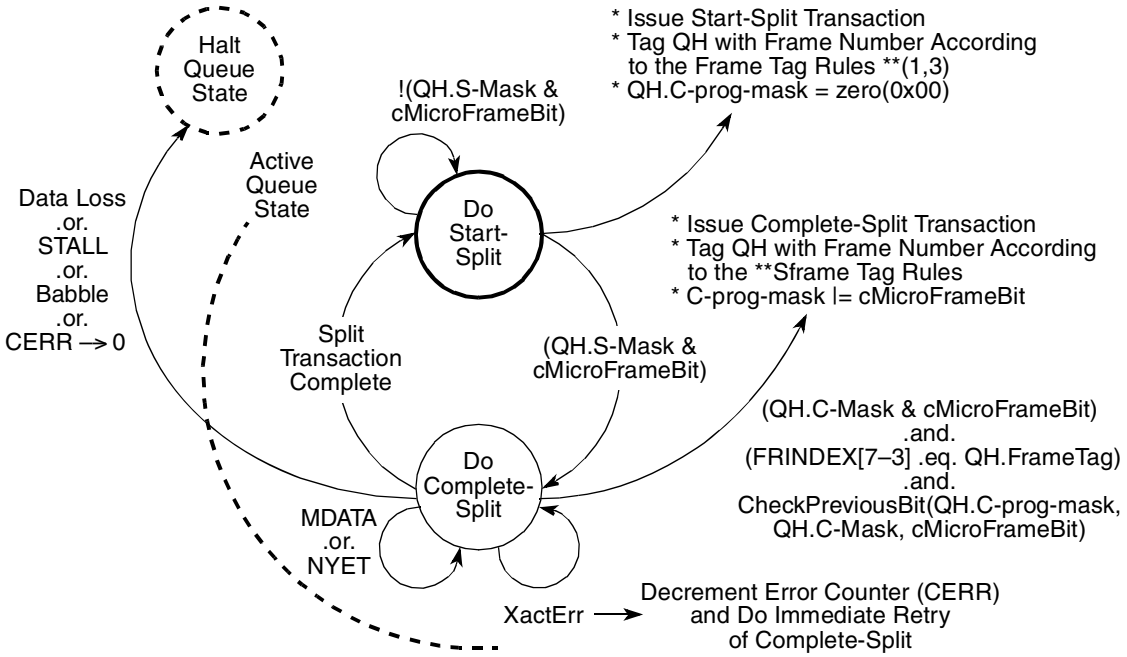


Figure 16-58. Split Transaction State Machine for Interrupt

16.6.12.2.6 Periodic Interrupt—Do-Start-Split

This is the state software must initialize a full- or low-speed interrupt queue head StartXState bit. This state is entered from the Do_Complete Split state only after the split transaction is complete. This occurs when one of the following events occur: The transaction translator responds to a complete-split transaction with one of the following:

- NAK. A NAK response is a propagation of the full- or low-speed endpoint's NAK response.
- ACK. An ACK response is a propagation of the full- or low-speed endpoint's ACK response. Only occurs on an OUT endpoint.
- DATA 0/1. Only occurs for INs. Indicates that this is the last of the data from the endpoint for this split transaction.
- ERR. The transaction on the low-/full-speed link below the transaction translator had a failure (for example, timeout, bad CRC, etc.).
- NYET (and Last). The host controller issued the last complete-split and the transaction translator responded with a NYET handshake. This means that the start-split was not correctly received by the transaction translator, so it never executed a transaction to the full- or low-speed endpoint, see Section Periodic Interrupt - Do Complete Split for the definition of 'Last'.

Each time the host controller visits a queue head in this state (once within the Execute Transaction state), bit-wise ANDs QH[S-mask] with cMicroFrameBit to determine whether to execute a start-split. If the result is non-zero, then the host controller issues a start-split transaction. If the PID Code field indicates an IN transaction, the host controller must zero-out the QH[S-bytes] field. After the split-transaction has been executed, the host controller sets up state in the queue head to track the progress of the complete-split phase of the split transaction. Specifically, it records the expected frame number into QH[FrameTag] field,

sets C-prog-mask to zero (0x00), and exits this state. Note that the host controller must not adjust the value of Cerr as a result of completion of a start-split transaction.

16.6.12.2.7 Periodic Interrupt—Do-Complete-Split

This state is entered unconditionally from the Do Start Split state after a start-split transaction is executed on the bus. Each time the host controller visits a queue head in this state (once within the Execute Transaction state), it checks to determine whether a complete-split transaction should be executed now.

There are four tests to determine whether a complete-split transaction should be executed.

- Test A. cMicroFrameBit is bit-wise ANDed with QH[C-mask] field. A non-zero result indicates that software scheduled a complete-split for this endpoint, during this micro-frame.
- Test B. QH[FrameTag] is compared with the current contents of FRINDEX[7–3]. An equal indicates a match.
- Test C. The complete-split progress bit vector is checked to determine whether the previous bit is set, indicating that the previous complete-split was appropriately executed. An example algorithm for this test is provided below:

```
Algorithm Boolean CheckPreviousBit(QH.C-prog-mask, QH.C-mask, cMicroFrameBit)
Begin
-- Return values:
-- TRUE - no error
-- FALSE - error
--
Boolean rvalue = TRUE;
previousBit = cMicroframeBit logical-rotate-right(1)
-- Bit-wise anding previousBit with C-mask indicates
-- whether there was an intent
-- to send a complete split in the previous micro-frame. So,
-- if the
-- 'previous bit' is set in C-mask, check C-prog-mask to
-- make sure it
-- happened.
If (previousBit bitAND QH.C-mask)then
                                If not(previousBit bitAND QH.C-prog-mask) then
         rvalue = FALSE;
     End if
End If
-- If the C-prog-mask already has a one in this bit position,
-- then an aliasing
-- error has occurred. It will probably get caught by the
-- FrameTag Test, but
-- at any rate it is an error condition that as detectable here
-- should not allow
-- a transaction to be executed.
If (cMicroFrameBit bitAND QH.C-prog-mask) then
rvalue = FALSE;
End if
return (rvalue)
End Algorithm
```

- Test D. Check to see if a start-split should be executed in this micro-frame. Note this is the same test performed in the Do Start Split state. Whenever it evaluates to TRUE and the controller is NOT processing in the context of a Recovery Path mode, it means a start-split should occur in this

micro-frame. Test D and Test A evaluating to TRUE at the same time is a system software error. Behavior is undefined.

If (A .and. B .and. C .and. not(D)) then the host controller will execute a complete-split transaction. When the host controller commits to executing the complete-split transaction, it updates QH[C-prog-mask] by bit-ORing with cMicroFrameBit. On completion of the complete-split transaction, the host controller records the result of the transaction in the queue head and sets QH[FrameTag] to the expected H-Frame number. The effect to the state of the queue head and thus the state of the transfer depends on the response by the transaction translator to the complete-split transaction. The following responses have the effects (note that any responses that result in decrementing of the Cerr will result in the queue head being halted by the host controller if the result of the decrement is zero):

- NYET (and Last). On each NYET response, the host controller checks to determine whether this is the last complete-split for this split transaction. Last is defined in this context as the condition where all of the scheduled complete-splits have been executed. If it is the last complete-split (with a NYET response), then the transfer state of the queue head is not advanced (never received any data) and this state exited. The transaction translator must have responded to all the complete-splits with NYETs, meaning that the start-split issued by the host controller was not received. The start-split should be retried at the next poll period.
- The test for whether this is the Last complete split can be performed by XOR QH[C-mask] with QH[C-prog-mask]. If the result is all zeros then all complete-splits have been executed. When this condition occurs, the XactErr status bit is set and the Cerr field is decremented.
- NYET (and not Last). See above description for testing for Last. The complete-split transaction received a NYET response from the transaction translator. Do not update any transfer state (except for C-prog-mask and FrameTag) and stay in this state. The host controller must not adjust Cerr on this response.
- Transaction Error (XactErr). Timeout, data CRC failure, etc. The Cerr field is decremented and the XactErr bit in the Status field is set. The complete split transaction is immediately retried (if Cerr is non-zero).If there is not enough time in the micro-frame to complete the retry and the endpoint is an IN, or Cerr is decremented to a zero from a one, the queue is halted. If there is not enough time in the micro-frame to complete the retry and the endpoint is an OUT and Cerr is not zero, then this state is exited (that is, return to Do Start Split). This results in a retry of the entire OUT split transaction, at the next poll period. Refer to Chapter 11 Hubs (specifically the section on full- and low-speed interrupts) in the *USB Specification Revision 2.0* for detailed requirements on why these errors must be immediately retried.
- ACK. This can only occur if the target endpoint is an OUT. The target endpoint ACK'd the data and this response is a propagation of the endpoint ACK up to the host controller. The host controller must advance the state of the transfer. The Current Offset field is incremented by Maximum Packet Length or Bytes to Transfer, whichever is less. The field Bytes To Transfer is decremented by the same amount. And the data toggle bit (dt) is toggled. The host controller will then exit this state for this queue head. The host controller must reload Cerr with maximum value on this response. Advancing the transfer state may cause other process events such as retirement of the qTD and advancement of the queue.

- MDATA. This response will only occur for an IN endpoint. The transaction translator responded with zero or more bytes of data and an MDATA PID. The incremental number of bytes received is accumulated in QH[S-bytes]. The host controller must not adjust Cerr on this response.
- DATA0/1. This response may only occur for an IN endpoint. The number of bytes received is added to the accumulated byte count in QH[S-bytes]. The state of the transfer is advanced by the result and the host controller exits this state for this queue head.
- Advancing the transfer state may cause other processing events such as retirement of the qTD and advancement of the queue.
- If the data sequence PID does not match the expected, the entirety of the data received in this split transaction is ignored, the transfer state is not advanced and this state is exited.
- NAK. The target endpoint Nak'd the full- or low-speed transaction. The state of the transfer is not advanced, and this state is exited. The host controller must reload Cerr with maximum value on this response.
- ERR. There was an error during the full- or low-speed transaction. The ERR status bit is set, Cerr is decremented, the state of the transfer is not advanced, and this state is exited.
- STALL. The queue is halted (an exit condition of the Execute Transaction state). The status field bits: Active bit is cleared and the Halted bit is set and the qTD is retired. Responses which are not enumerated in the list or which are received out of sequence are illegal and may result in undefined host controller behavior. The other possible combinations of tests A, B, C, and D may indicate that data or response was lost. Table 16-74 lists the possible combinations and the appropriate action.

Table 16-74. 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 micro-frame.Host controller should continue walking the schedule.
A not(C)	If PIDCode = IN Halt QHDIf PIDCode = OUT Retry start-split	Progress bit check failed. These means a complete-split has been missed. There is the possibility of lost data. If PID Code is an IN, then the Queue head must be halted.If PID Code is an OUT, then the transfer state is not advanced and the state exited (for example, start-split is retried). This is a host-induced error and does not effect Cerr. In either case, set the Missed Micro-frame bit in the status field to a one.
A 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 Micro-frame bit in the status field to a one.

Table 16-74. Interrupt IN/OUT Do Complete Split State Execution Criteria (continued)

Condition	Action	Description
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 QHDIf 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 Micro-frame bit in the status field to a one. Note that when executing in the context of a Recovery Path mode, the host controller is allowed to process the queue head and take the actions indicated above, or it may wait until the queue head is visited in the normal processing mode. Regardless, the host controller must not execute a start-split in the context of a executing in a Recovery Path mode.

16.6.12.2.8 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 16-55).
- 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 16-55.
- 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 16-55.

16.6.12.2.9 Rebalancing the Periodic Schedule

System software must occasionally adjust a periodic queue head's S-mask and C-mask fields during operation. This need occurs when adjustments to the periodic schedule create a new bandwidth budget and one or more queue head's are assigned new execution footprints (that is, new S-mask and C-mask values).

It is imperative that system software must not update these masks to new values in the midst of a split transaction. In order to avoid any race conditions with the update, the host controller provides a simple assist to system software. System software sets the Inactivate-on-next-Transaction (I) bit to signal the host controller that it intends to update the S-mask and C-mask on this queue head. System software then waits for the host controller to observe the I-bit is set and transitions the Active bit to a zero. The rules for how and when the host controller clears the Active bit are:

- If the Active bit is cleared, no action is taken. The host controller does not attempt to advance the queue when the I-bit is set.

- If the Active bit is set and the SplitXState is DoStart (regardless of the value of S-mask), the host controller simply clears the Active bit. The host controller is not required to write the transfer state back to the current qTD. Note that if the S-mask indicates that a start-split is scheduled for the current micro-frame, the host controller must not issue the start-split bus transaction; it must clear the Active bit.

System software must save transfer state before setting the I-bit. This is required so that it can correctly determine what transfer progress (if any) occurred after the I-bit was set and the host controller executed it's final bus-transaction and cleared the Active bit.

After system software has updated the S-mask and C-mask, it must then reactivate the queue head. Since the Active bit and the I-bit cannot be updated with the same write, system software needs to use the following algorithm to coherently re-activate a queue head that has been stopped using the I-bit.

1. Set the Halted bit, then
2. Clear the I-bit, then
3. Set the Active bit and clear the Halted bit in the same write.

Setting the Halted bit inhibits the host controller from attempting to advance the queue between the time the I-bit is cleared and the Active bit is set.

16.6.12.3 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 16.6.8, “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.

16.6.12.3.1 Split Transaction Scheduling Mechanisms for Isochronous

Full-speed isochronous transactions are managed through a transaction translator's periodic pipeline. As with full- and low-speed interrupt, system software manages each transaction translator's periodic pipeline by budgeting and scheduling exactly during which micro-frames the start-splits and complete-splits for each full-speed isochronous endpoint occur. The requirements described in Section Split Transaction Scheduling Mechanisms for Interrupt apply. Figure 16-59 illustrates the general scheduling boundary conditions that are supported by the EHCI periodic schedule. The S_n and C_n labels indicate micro-frames where software can schedule start- and complete-splits (respectively). The H-Frame boundaries are marked with a large, solid bold vertical line. The B-Frame boundaries are marked with a large, bold, dashed line. The bottom of the figure illustrates the relationship of an siTD to the H-Frame.

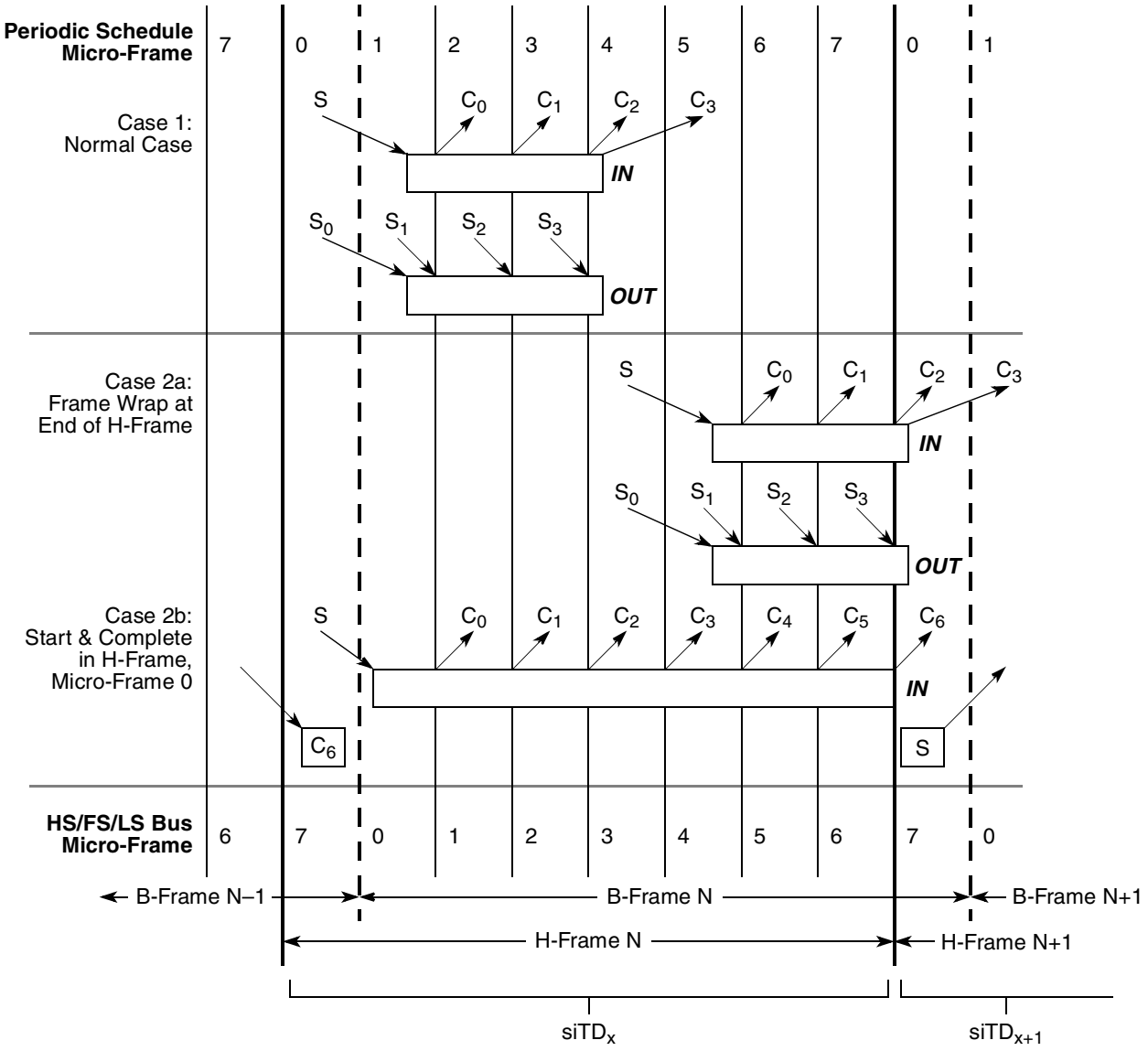


Figure 16-59. Split Transaction, Isochronous Scheduling Boundary Conditions

When the endpoint is an isochronous OUT, there are only start-splits, and no complete-splits. When the endpoint is an isochronous IN, there is at most one start-split and one to N complete-splits. The scheduling boundary cases are:

- Case 1: The entire split transaction is completely bounded by an H-Frame. For example, the start-splits and complete-splits are all scheduled to occur in the same H-Frame.
- Case 2a: This boundary case is where one or more (at most two) complete-splits of a split transaction IN are scheduled across an H-Frame boundary. This can only occur when the split transaction has the possibility of moving data in B-Frame, micro-frames 6 or 7 (H-Frame micro-frame 7 or 0). When an H-Frame boundary wrap condition occurs, the scheduling of the split transaction spans more than one location in the periodic list.(for example, it takes two siTDs in adjacent periodic frame list locations to fully describe the scheduling for the split transaction).

Although the scheduling of the split transaction may take two data structures, all of the complete-splits for each full-speed IN isochronous transaction must use only one data pointer. For this reason, siTDs contain a back pointer.

Software must never schedule full-speed isochronous OUTs across an H-Frame boundary.

- Case 2b: This case can only occur for a very large isochronous IN. It is the only allowed scenario where a start-split and complete-split for the same endpoint can occur in the same micro-frame. Software must enforce this rule by scheduling the large transaction first. Large is defined to be anything larger than 579 byte maximum packet size.

A subset of the same mechanisms employed by full- and low-speed interrupt queue heads are employed in siTDs to schedule and track the portions of isochronous split transactions. The following fields are initialized by system software to instruct the host controller when to execute portions of the split transaction protocol:

- SplitXState. This is a single bit residing in the Status field of an siTD (see Table 16-54). This bit is used to track the current state of the split transaction. The rules for managing this bit are described in Section 16.6.12.3.3, “Split Transaction Execution State Machine for Isochronous.”
- Frame S-mask. This is a bit-field wherein system software sets a bit corresponding to the micro-frame (within an H-Frame) that the host controller should execute a start-split transaction. This is always qualified by the value of the SplitXState bit. For example, referring to the IN example in Figure 16-59, case 1, the S-mask would have a value of 0b0000_0001 indicating that if the siTD is traversed by the host controller, and the SplitXState indicates Do Start Split, and the current micro-frame as indicated by FRINDEX[2–0] is 0, then execute a start-split transaction.
- Frame C-mask. This is a bit-field where system software sets one or more bits corresponding to the micro-frames (within an H-Frame) that the host controller should execute complete-split transactions. The interpretation of this field is always qualified by the value of the SplitXState bit. For example, referring to the IN example in Figure 16-59, case 1, the C-mask would have a value of 0b 0011_1100 indicating that if the siTD is traversed by the host controller, and the SplitXState indicates Do Complete Split, and the current micro-frame as indicated by FRINDEX[2–0] is 2, 3, 4, or 5, then execute a complete-split transaction.
- Back Pointer. This field in a siTD is used to complete an IN split-transaction using the previous H-Frame's siTD. This is only used when the scheduling of the complete-splits span an H-Frame boundary.

There exists a one-to-one relationship between a high-speed isochronous split transaction (including all start- and complete-splits) and one full-speed isochronous transaction. An siTD contains (amongst other things) buffer state and split transaction scheduling information. An siTD's buffer state always maps to one full-speed isochronous data payload. This means that for any full-speed transaction payload, a single siTD's data buffer must be used. This rule applies to both IN an OUTs. An siTD's scheduling information usually also maps to one high-speed isochronous split transaction. The exception to this rule is the H-Frame boundary wrap cases mentioned above.

The siTD data structure describes at most, one frame's worth of high-speed transactions and that description is strictly bounded within a frame boundary. Figure 16-60 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.

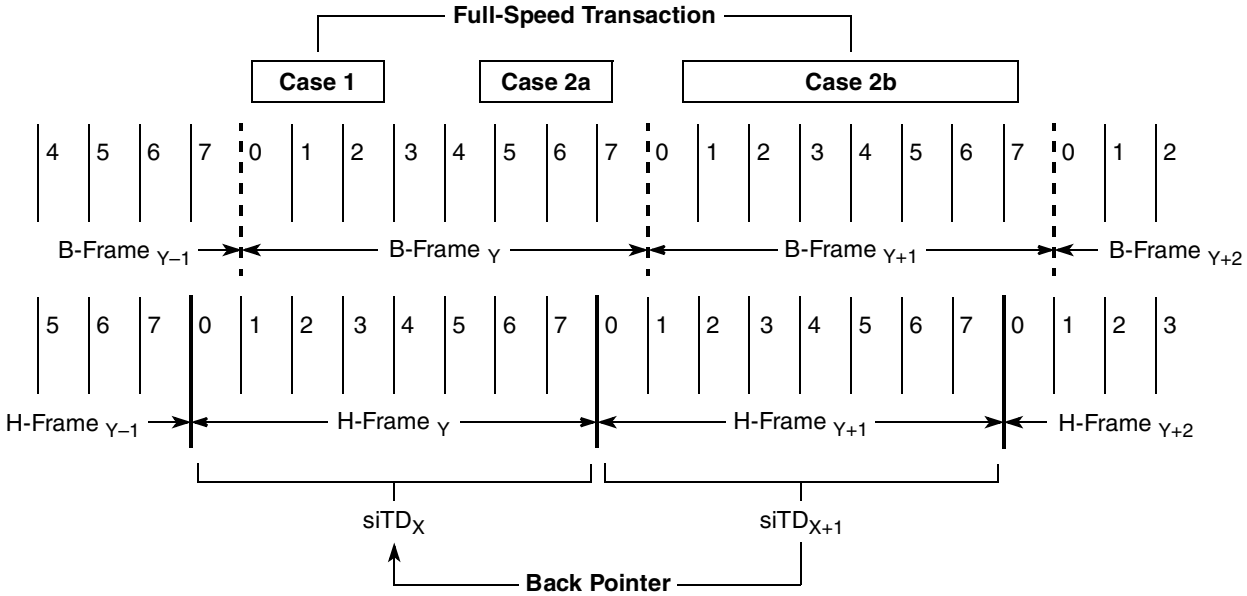


Figure 16-60. siTD Scheduling Boundary Examples

Each case is described below:

- Case 1: One siTD is sufficient to describe and complete the isochronous split transaction because the whole isochronous split transaction is tightly contained within a single H-Frame.
- Case 2a, 2b: Although both INs and OUTs can have these footprints, OUTs always take only one siTD to schedule. However, INs (for these boundary cases) require two siTDs to complete the scheduling of the isochronous split transaction. siTDX is used to always issue the start-split and the first N complete-splits. The full-speed transaction (for these cases) can deliver data on the full-speed bus segment during micro-frame 7 of H-Frame$_{Y+1}$, or micro-frame 0 of H-Frame$_{Y+2}$. The complete splits are scheduled using siTD$_{X+2}$ (not shown). The complete-splits to extract this data must use the buffer pointer from siTD$_{X+1}$. The only way for the host controller to reach siTD$_{X+1}$ from H-Frame$_{Y+2}$ is to use siTD$_{X+2}$'s back pointer.

Software must apply the following rules when calculating the schedule and linking the schedule data structures into the periodic schedule:

- Software must ensure that an isochronous split-transaction is started so that it will complete before the end of the B-Frame.
- Software must ensure that for a single full-speed isochronous endpoint, there is never a start-split and complete-split in H-Frame, micro-frame 1. This is mandated as a rule so that case 2a and case 2b can be discriminated. According to the core USB specification, the long isochronous transaction illustrated in Case 2b, could be scheduled so that the start-split was in micro-frame 1 of H-Frame N and the last complete-split would need to occur in micro-frame 1 of H-Frame N+1. However, it is impossible to discriminate between cases 2a and case 2b, which has significant impact on the complexity of the host controller.

16.6.12.3.2 Tracking Split Transaction Progress for Isochronous Transfers

Isochronous endpoints do not employ the concept of a halt on error, however the host controller does identify and report per-packet errors observed in the data stream. This includes schedule traversal problems (skipped micro-frames), timeouts and corrupted data received.

In similar kind to interrupt split-transactions, the portions of the split transaction protocol must execute in the micro-frames they are scheduled. The queue head data structure used to manage full- and low-speed interrupt has several mechanisms for tracking when portions of a transaction have occurred. Isochronous transfers use siTDs for their transfers and the data structures are only reachable using the schedule in the exact micro-frame in which they are required (so all the mechanism employed for tracking in queue heads is not required for siTDs). Software has the option of reusing siTD several times in the complete periodic schedule. However, it must ensure that the results of split transaction N are consumed and the siTD re-initialized (activated) before the host controller gets back to the siTD (in a future micro-frame).

Split-transaction isochronous OUTs utilize a low-level protocol to indicate which portions of the split transaction data have arrived. Control over the low-level protocol is exposed in an siTD using the fields Transaction Position (TP) and Transaction Count (T-count). If the entire data payload for the OUT split transaction is larger than 188 bytes, there will be more than one start-split transaction, each of which require proper annotation. If host hold-offs occur, then the sequence of annotations received from the host will not be complete, which is detected and handled by the transaction translator. See Section 16.6.12.3.1, “Split Transaction Scheduling Mechanisms for Isochronous,” for a description on how these fields are used during a sequence of start-split transactions.

The fields siTD[T-Count] and siTD[TP] are used by the host controller to drive and sequence the transaction position annotations. It is the responsibility of system software to properly initialize these fields in each siTD. Once the budget for a split-transaction isochronous endpoint is established, S-mask, T-Count, and TP initialization values for all the siTD associated with the endpoint are constant. They remain constant until the budget for the endpoint is recalculated by software and the periodic schedule adjusted.

For IN-endpoints, the transaction translator simply annotates the response data packets with enough information to allow the host controller to identify the last data. As with split transaction Interrupt, it is the host controller's responsibility to detect when it has missed an opportunity to execute a complete-split. The following field in the siTD is used to track and detect errors in the execution of a split transaction for an IN isochronous endpoint.

- C-prog-mask. This is an eight-bit bit-vector where the host controller keeps track of which complete-splits have been executed. Due to the nature of the Transaction Translator periodic pipeline, the complete-splits need to be executed in-order. The host controller needs to detect when the complete-splits have not been executed in order. This can only occur due to system hold-offs where the host controller cannot get to the memory-based schedule. C-prog-mask is a simple bit-vector that the host controller sets a bit for each complete-split executed. The bit position is determined by the micro-frame (FRINDEX[2–0]) number in which the complete-split was executed. The host controller always checks C-prog-mask before executing a complete-split transaction. If the previous complete-splits have not been executed, then it means one (or more) have been skipped and data has potentially been lost. System software is required to initialize this field to zero before setting an siTD's Active bit to a one.

If a transaction translator returns with the final data before all of the complete-splits have been executed, the state of the transfer is advanced so that the remaining complete-splits are not executed. It is important to note that an IN siTD is retired based solely on the responses from the Transaction Translator to the complete-split transactions. This means, for example, that it is possible for a transaction translator to respond to a complete-split with an MDATA PID. The number of bytes in the MDATA's data payload could cause the siTD[Total Bytes to Transfer] field to decrement to zero. This response can occur, before all of the scheduled complete-splits have been executed. In other interface, data structures (for example, high-speed data streams through queue heads), the transition of Total Bytes to Transfer to zero signals the end of the transfer and results in clearing the Active bit. However, in this case, the result has not been delivered by the Transaction Translator and the host must continue with the next complete-split transaction to extract the residual transaction state. This scenario occurs because of the pipeline rules for a Transaction Translator. In summary, the periodic pipeline rules require that on a micro-frame boundary, the Transaction Translator holds the final two bytes received (if it has not seen an End Of Packet (EOP)) in the full-speed bus pipe stage and gives the remaining bytes to the high-speed pipeline stage. At the micro-frame boundary, the Transaction Translator could have received the entire packet (including both CRC bytes) but not received the packet EOP. In the next micro-frame, the Transaction Translator responds with an MDATA and sends all of the data bytes (with the two CRC bytes being held in the full-speed pipeline stage). This could cause the siTD to decrement it's Total Bytes to Transfer field to zero, indicating it has received all expected data. The host must still execute one more (scheduled) complete-split transaction in order to extract the results of the full-speed transaction from the Transaction Translator (for example, the Transaction Translator may have detected a CRC failure, and this result must be forwarded to the host).

If the host experiences hold-offs that cause the host controller to skip one or more (but not all) scheduled split transactions for an isochronous OUT, then the protocol to the transaction translator is not consistent and the transaction translator detects and reacts to the problem. Likewise, for host hold-offs that cause the host controller to skip one or more (but not all) scheduled split transactions for an isochronous IN, the C-prog-mask is used by the host controller to detect errors. However, if the host experiences a hold-off that causes it to skip all of an siTD, or an siTD expires during a host hold off (for example, a hold-off occurs and the siTD is no longer reachable by the host controller in order for it to report the hold-off event), then system software must detect that the siTDs have not been processed by the host controller (for example, state not advanced) and report the appropriate error to the client driver.

16.6.12.3.3 Split Transaction Execution State Machine for Isochronous

In this section, all references to micro-frame are in the context of a micro-frame within an H-Frame.

If the Active bit in the Status byte is a zero, the host controller ignores the siTD and continues traversing the periodic schedule. Otherwise the host controller processes the siTD as specified below. A split transaction state machine is used to manage the split-transaction protocol sequence. The host controller uses the fields defined in Section 16.6.12.3.2, “Tracking Split Transaction Progress for Isochronous Transfers,” plus the variable cMicroFrameBit defined in Section 16.6.12.2.5, “Split Transaction Execution State Machine for Interrupt,” to track the progress of an isochronous split transaction. Figure 16-61 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.

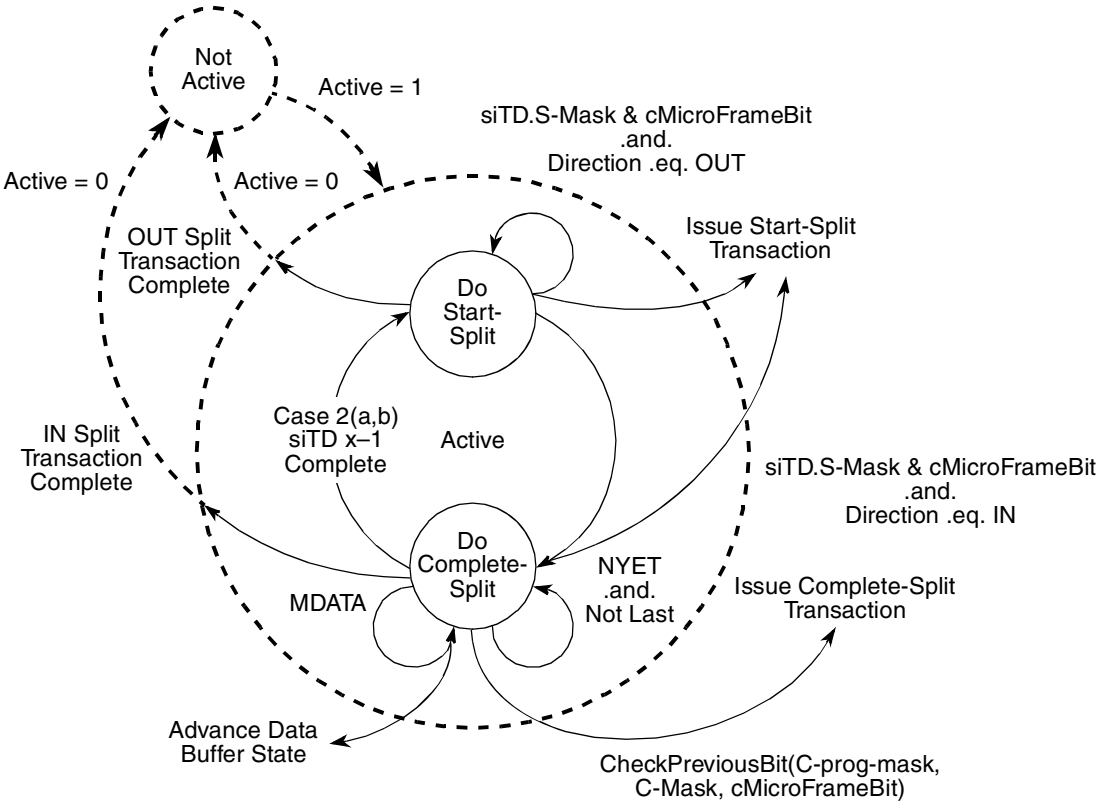


Figure 16-61. Split Transaction State Machine for Isochronous

16.6.12.3.4 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 16-60) is used to determine the initial value of TP. The initial cases are summarized in Table 16-75.

Table 16-75. Initial Conditions for OUT siTD's 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 16-76 illustrates all of the TP and T-count transitions, which must be accomplished by the host controller.

Table 16-76. 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 16-76.

These fields are then written back to the memory based siTD. The S-mask is fixed for the life of the current budget. As mentioned above, TP and T-count are set specifically in each siTD to reflect the data to be sent from this siTD. Therefore, regardless of the value of S-mask, the actual number of start-split transactions depends on T-count (or equivalently, Total Bytes to Transfer). The host controller must clear the Active bit when it detects that all of the schedule data has been sent to the bus. The preferred method is to detect when T-Count decrements to zero as a result of a start-split bus transaction. Equivalently, the host controller can detect when Total Bytes to Transfer decrements to zero. Either implementation must ensure that if the

initial condition is Total Bytes to Transfer is equal to zero and T-count is equal to a one, then the host controller will issue a single start-split, with a zero-length data payload. Software must ensure that TP, T-count and Total Bytes to Transfer are set to deliver the appropriate number of bus transactions from each siTD. An inconsistent combination will yield undefined behavior.

If the host experiences hold-offs that cause the host controller to skip start-split transactions for an OUT transfer, the state of the transfer will not progress appropriately. The transaction translator observes protocol violations in the arrival of the start-splits for the OUT endpoint (that is, the transaction position annotation is incorrect as received by the transaction translator).

Example scenarios are described in Section 16.6.12.3.7, “Split Transaction for Isochronous - Processing Examples.”

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 16.6.12.3.5, “Periodic Isochronous - Do Complete Split,” can be used prior to executing each start-split to determine whether start-splits were skipped. The host controller can use this mechanism to detect missed micro-frames. It can then clear the siTD's Active bit and stop execution of this siTD. This saves on both memory and high-speed bus bandwidth.

16.6.12.3.5 Periodic Isochronous - Do Complete Split

This state is only used by a split-transaction isochronous IN endpoint. This state is entered unconditionally from the Do Start State after a start-split transaction is executed for an IN endpoint. Each time the host controller visits an siTD in this state, it conducts a number of tests to determine whether it should execute a complete-split transaction. The individual tests are listed below. The sequence they are applied depends on which micro-frame the host controller is currently executing which means that the tests might not be applied until after the siTD referenced from the back pointer has been fetched.

- Test A. cMicroFrameBit is bit-wise ANDed with the siTD[C-mask] field. A non-zero result indicates that software scheduled a complete-split for this endpoint, during this micro-frame. This test is always applied to a newly fetched siTD that is in this state.
- Test B. The siTD[C-prog-mask] bit vector is checked to determine whether the previous complete splits have been executed. An example algorithm is given below (this is slightly different than the algorithm used in Section 16.6.12.2.7, “Periodic Interrupt—Do-Complete-Split”). The sequence in which this test is applied depends on the current value of FRINDEX[2–0]. If FRINDEX[2–0] is 0 or 1, it is not applied until the back pointer has been used. Otherwise it is applied immediately.

```
Algorithm Boolean CheckPreviousBit(siTD.C-prog-mask, siTD.C-mask, cMicroFrameBit)
Begin
     Boolean rvalue = TRUE;
     previousBit = cMicroFrameBit rotate-right(1)
     -- Bit-wise anding previousBit with C-mask indicates whether there
     -- was an intent to send a complete split in the previous micro-
     -- frame. So, if the 'previous bit' is set in C-mask, check
     -- C-prog-mask to make sure it happened.
     if previousBit bitAND siTD.C-mask then
          if not (previousBit bitAND siTD.C-prog-mask) then
                                                              rvalue = FALSE
          End if
     End if
```

```
        Return rvalue
    End Algorithm
```

If Test A is true and FRINDEX[2–0] is zero or one, then this is a case 2a or 2b scheduling boundary (see Figure 16-59). See Section 16.6.12.3.6, “Complete-Split for Scheduling Boundary Cases 2a, 2b,” for details in handling this condition.

If Test A and Test B evaluate to true, then the host controller executes a complete-split transaction using the transfer state of the current siTD. When the host controller commits to executing the complete-split transaction, it updates QH[C-prog-mask] by bit-ORing with cMicroFrameBit. The transfer state is advanced based on the completion status of the complete-split transaction. To advance the transfer state of an IN siTD, the host controller must:

- Decrement the number of bytes received from siTD[Total Bytes To Transfer]
- Adjust siTD[Current Offset] by the number of bytes received
- Adjust the siTD[P] (page select) field if the transfer caused the host controller to use the next page pointer
- Set any appropriate bits in the siTD[Status] field, depending on the results of the transaction.

Note that if the host controller encounters a condition where siTD[Total Bytes To Transfer] is zero, and it receives more data, the host controller must not write the additional data to memory. The siTD[Status-Active] bit must be cleared and the siTD[Status-Babble Detected] bit must be set. The fields siTD[Total Bytes To Transfer], siTD[Current Offset], and siTD[P] are not required to be updated as a result of this transaction attempt.

The host controller accepts (assuming good data packet CRC and sufficient room in the buffer as indicated by the value of siTD[Total Bytes To Transfer]) MDATA and DATA0/1 data payloads up to and including 192 bytes. The host controller may optionally clear siTD[Status-Active] and set siTD[Status-Babble Detected] when it receives MDATA or DATA0/1 with a data payload of more than 192 bytes. The following responses have the noted effects:

- ERR. The full-speed transaction completed with a time-out or bad CRC and this is a reflection of that error to the host. The host controller sets the ERR bit in the siTD[Status] field and clears the Active bit.
- Transaction Error (XactErr). The complete-split transaction encounters a Timeout, CRC16 failure, etc. The siTD[Status] field XactErr field is set and the complete-split transaction must be retried immediately. The host controller must use an internal error counter to count the number of retries as a counter field is not provided in the siTD data structure. The host controller will not retry more than two times. If the host controller exhausts the retries or the end of the micro-frame occurs, the Active bit is cleared.
- DATAx (0 or 1). This response signals that the final data for the split transaction has arrived. The transfer state of the siTD is advanced and the Active bit is cleared. If the Bytes To Transfer field has not decremented to zero (including the reception of the data payload in the DATAx response), then less data than was expected, or allowed for was actually received. This short packet event does not set the USBINT status bit in the USBSTS register to a one. The host controller will not detect this condition.

- NYET (and Last). On each NYET response, the host controller also checks to determine whether this is the last complete-split for this split transaction. Last was defined in Section Periodic Interrupt - Do Complete Split. If it is the last complete-split (with a NYET response), then the transfer state of the siTD is not advanced (never received any data) and the Active bit is cleared. No bits are set in the Status field because this is essentially a skipped transaction. The transaction translator must have responded to all the scheduled complete-splits with NYETs, meaning that the start-split issued by the host controller was not received. This result should be interpreted by system software as if the transaction was completely skipped. The test for whether this is the last complete split can be performed by XORing C-mask with C-prog-mask. A zero result indicates that all complete-splits have been executed.
- MDATA (and Last). See above description for testing for Last. This can only occur when there is an error condition. Either there has been a babble condition on the full-speed link, which delayed the completion of the full-speed transaction, or software set up the S-mask and/or C-masks incorrectly. The host controller must set the XactErr bit and clear the Active bit.
- NYET (and not Last). See above description for testing for Last. The complete-split transaction received a NYET response from the transaction translator. Do not update any transfer state (except for C-prog-mask) and stay in this state.
- MDATA (and not Last). The transaction translator responds with an MDATA when it has partial data for the split transaction. For example, the full-speed transaction data payload spans from micro-frame X to X+1 and during micro-frame X, the transaction translator responds with an MDATA and the data accumulated up to the end of micro-frame X. The host controller advances the transfer state to reflect the number of bytes received.

If Test A succeeds, but Test B fails, it means that one or more of the complete-splits have been skipped. The host controller sets the Missed Micro-Frame status bit and clears the Active bit.

16.6.12.3.6 Complete-Split for Scheduling Boundary Cases 2a, 2b

Boundary cases 2a and 2b (INs only) (see Figure 16-59) require that the host controller use the transaction state context of the previous siTD to finish the split transaction. Table 16-77 enumerates the transaction state fields.

Table 16-77. 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 $siTD_X$[Back Pointer] T-bit is zero, or
- If cMicroFrameBit is a 0x2 and siTDX[S-mask[0]] is zero

When either of these conditions apply, then the host controller must use the transaction state from $siTD_{X-1}$.

In order to access $siTD_{X-1}$, the host controller reads on-chip the siTD referenced from $siTD_X$[Back Pointer].

The host controller must save the entire state from $siTD_X$ while processing $siTD_{X-1}$. This is to accommodate for case 2b processing. The host controller must not recursively walk the list of siTD[Back Pointers].

If $siTD_{X-1}$ is active (Active bit is set and SplitXStat is Do Complete Split), then both Test A and Test B are applied as described above. If these criteria to execute a complete-split are met, the host controller executes the complete split and evaluates the results as described above. The transaction state (see Table 16-77) of $siTD_{X-1}$ is appropriately advanced based on the results and written back to memory. If the resultant state of $siTD_{X-1}$'s Active bit is a one, then the host controller returns to the context of $siTD_X$, and follows its next pointer to the next schedule item. No updates to $siTD_X$ are necessary.

If $siTD_{X-1}$ is active (Active bit is set and SplitXStat is Do Start Split), then the host controller must clear the Active bit and set the Missed Micro-Frame status bit and the resultant status is written back to memory.

If $siTD_{X-1}$'s Active bit is cleared, (because it was cleared when the host controller first visited $siTD_{X-1}$ via $siTD_X$'s back pointer, it transitioned to zero as a result of a detected error, or the results of $siTD_{X-1}$'s complete-split transaction cleared it), then the host controller returns to the context of $siTD_X$ and transitions its SplitXState to Do Start Split. The host controller then determines whether the case 2b start split boundary condition exists (that is, if cMicroframeBit is 1 and $siTD_X$[S-mask[0]] is 1). If this criterion is met the host controller immediately executes a start-split transaction and appropriately advances the transaction state of $siTD_X$, then follows $siTD_X$[Next Pointer] to the next schedule item. If the criterion is not met, the host controller simply follows $siTD_X$[Next Pointer] to the next schedule item. Note that in the case of a 2b boundary case, the split-transaction of $siTD_{X-1}$ will have its Active bit cleared when the host controller returns to the context of $siTD_X$. Also, note that software should not initialize an siTD with C-mask bits 0 and 1 set and an S-mask with bit 0 set. This scheduling combination is not supported and the behavior of the host controller is undefined.

16.6.12.3.7 Split Transaction for Isochronous - Processing Examples

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 16-78 illustrates a few frames worth of scheduling required to schedule a case 2a full-speed isochronous data stream.

Table 16-78. Example Case 2a - Software Scheduling siTDs for an IN Endpoint

siTDX		Micro-Frames								InitialSplitXState
#	Masks	0	1	2	3	4	5	6	7	
X	S-Mask					1				Do Start Split
	C-Mask	1	1					1	1	
X+1	S-Mask					1				Do Complete Split
	C-Mask	1	1					1	1	
X+2	S-Mask					1				Do Complete Split
	C-Mask	1	1					1	1	
X+3	S-Mask	Repeats previous pattern								Do Complete Split
	C-Mask									

This example shows the first three siTDs for the transaction stream. Since this is the case-2a frame-wrap case, S-masks of all siTDs for this endpoint have a value of 0x10 (a one bit in micro-frame 4) and C-mask value of 0xC3 (one-bits in micro-frames 0,1, 6 and 7). Additionally, software ensures that the Back Pointer field of each siTD references the appropriate siTD data structure (and the Back Pointer T-bits are cleared).

The initial SplitXState of the first siTD is Do Start Split. The host controller will visit the first siTD eight times during frame X. The C-mask bits in micro-frames 0 and 1 are ignored because the state is Do Start Split. During micro-frame 4, the host controller determines that it can run a start-split (and does) and changes SplitXState to Do Complete Split. During micro-frames 6 and 7, the host controller executes complete-splits. Notice the siTD for frame X+1 has it's SplitXState initialized to Do Complete Split. As the host controller continues to traverse the schedule during H-Frame X+1, it will visit the second siTD eight times. During micro-frames 0 and 1 it will detect that it must execute complete-splits.

During H-Frame X+1, micro-frame 0, the host controller detects that $siTD_{X+1}$'s Back Pointer[T] bit is a zero, saves the state of $siTD_{X+1}$ and fetches $siTD_X$. It executes the complete split transaction using the transaction state of $siTD_X$. If the $siTD_X$ split transaction is complete, siTD's Active bit is cleared and results written back to $siTD_X$. The host controller retains the fact that $siTD_X$ is retired and transitions the SplitXState in $siTD_{X+1}$ to Do Start Split. At this point, the host controller is prepared to execute the start-split for $siTD_{X+1}$ when it reaches micro-frame 4. If the split-transaction completes early (transaction-complete is defined in Section 16.6.12.3.5, “Periodic Isochronous - Do Complete Split”), that is, before all the scheduled complete-splits have been executed, the host controller changes $siTD_X$[SplitXState] to Do Start Split early and naturally skips the remaining scheduled complete-split

transactions. For this example, $siTD_{X+1}$ does not receive a DATA0 response until H-Frame X+2, micro-frame 1.

During H-Frame X+2, micro-frame 0, the host controller detects that $siTD_{X+2}$'s Back Pointer[T] bit is zero, saves the state of $siTD_{X+2}$ and fetches $siTD_{X+1}$. As described above, it executes another split transaction, receives an MDATA response, updates the transfer state, but does not modify the Active bit. The host controller returns to the context of $siTD_{X+2}$, and traverses it's next pointer without any state change updates to $siTD_{X+2}$.

During H-Frame X+2, micro-frame 1, the host controller detects $siTD_{X+2}$'s S-mask[0] bit is zero, saves the state of $siTD_{X+2}$ and fetches $siTD_{X+1}$. It executes another complete-split transaction, receives a DATA0 response, updates the transfer state and clears the Active bit. It returns to the state of $siTD_{X+2}$ and changes its SplitXState to Do Start Split. At this point, the host controller is prepared to execute start-splits for $siTD_{X+2}$ when it reaches micro-frame 4.

16.6.13 Port Test Modes

EHCI host controllers implement the port test modes Test J_State, Test K_State, Test_Packet, Test Force_Enable, and Test SE0_NAK as described in the *USB Specification Revision 2.0*. The required, port test sequence is (assuming the CF-bit in the CONFIGFLAG register is set):

- Disable the periodic and asynchronous schedules by clearing the Asynchronous Schedule Enable and Periodic Schedule Enable bits in the USBCMD register.
- Place all enabled root ports into the suspended state by setting the Suspend bit in each appropriate PORTSC register.
- Clear the Run/Stop bit in the USBCMD register and wait for the HCHalted bit in the USBSTS register, to transition to a one. Note that an EHCI host controller implementation may optionally allow port testing with the Run/Stop bit set. However, all host controllers must support port testing with Run/Stop cleared and HCHalted set.
- Set the Port Test Control field in the port under test PORTSC register to the value corresponding to the desired test mode. If the selected test is Test_Force_Enable, then the Run/Stop bit in the USBCMD register must then be transitioned back to one, in order to enable transmission of SOFs out of the port under test.
- When the test is complete, system software must ensure the host controller is halted (HCHalted bit is a one) then it terminates and exits test mode by setting HCReset.

16.6.14 Interrupts

The EHCI host controller hardware provides interrupt capability based on a number of sources. There are several general groups of interrupt sources:

- Interrupts as a result of executing transactions from the schedule (success and error conditions),
- Host controller events (Port change events, etc.), and
- Host controller error events

All transaction-based sources are maskable through the host controller's Interrupt Enable register (USBINTR). Additionally, individual transfer descriptors can be marked to generate an interrupt on completion. This section describes each interrupt source and the processing that occurs in response to the interrupt.

During normal operation, interrupts may be immediate or deferred until the next interrupt threshold occurs. The interrupt threshold is a tunable parameter via the Interrupt Threshold Control field in the USBCMD register. The value of this register controls when the host controller generates an interrupt on behalf of normal transaction execution. When a transaction completes during an interrupt interval period, the interrupt signaling the completion of the transfer will not occur until the interrupt threshold occurs. For example, the default value is eight micro-frames. This means that the host controller will not generate interrupts any more frequently than once every eight micro-frames.

Section 16.6.14.2.4, “Host System Error” details effects of a host system error.

If an interrupt has been scheduled to be generated for the current interrupt threshold interval, the interrupt is not signaled until after the status for the last complete transaction in the interval has been written back to system memory. This may sometimes result in the interrupt not being signaled until the next interrupt threshold.

Initial interrupt processing is the same, regardless of the reason for the interrupt. When an interrupt is signaled by the hardware, CPU control is transferred to host controller's USB interrupt handler. The precise mechanism to accomplish the transfer is OS specific. For this discussion it is just assumed that control is received. When the interrupt handler receives control, its first action is to reads the USBSTS. It then acknowledges the interrupt by clearing all of the interrupt status bits by writing ones to these bit positions. The handler then determines whether the interrupt is due to schedule processing or some other event. After acknowledging the interrupt, the handler (via an OS-specific mechanism), schedules a deferred procedure call (DPC) which will execute later. The DPC routine processes the results of the schedule execution. The precise mechanisms used are beyond the scope of this document.

NOTE

The only method software should use for acknowledging an interrupt is by transitioning the appropriate status bits in the USBSTS register from a one to a zero.

16.6.14.1 Transfer/Transaction Based Interrupts

These interrupt sources are associated with transfer and transaction progress. They are all dependent on the next interrupt threshold.

16.6.14.1.1 Transaction Error

A transaction error is any error that caused the host controller to think that the transfer did not complete successfully. Table 16-79 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 16-79. Summary of Transaction Errors

Event/ Result	Queue Head/qTD/iTD/siTD Side Effects		USBSTS[USBERRINT]
	Cerr	Status Field	
CRC	-1	XactErr set	1[1]
Timeout	-1	XactErr set	1[1]
Bad PID[2]	-1	XactErr set	1[1]
Babble	N/A	See Section 16.6.14.1.2, “Serial Bus Babble”	1
Buffer Error	N/A	See Section 16.6.14.1.3, “Data Buffer Error”	

1 If occurs in a queue head, then USBERRINT is asserted only when Cerr counts down from a one to a zero. In addition the queue is halted.

2 The host controller received a response from the device, but it could not recognize the PID as a valid PID.

There is a small set of protocol errors that relate only when executing a queue head and fit under the umbrella of a WRONG PID error that are significant to explicitly identify. When these errors occur, the XactErr status bit in the queue head is set and the Cerr field is decremented. When the PID Code indicates a SETUP, the following responses are protocol errors and result in XactErr bit being set and the Cerr field being decremented.

- EPS field indicates a high-speed device and it returns a Nak handshake to a SETUP.
- EPS field indicates a high-speed device and it returns a Nyet handshake to a SETUP.
- EPS field indicates a low- or full-speed device and the complete-split receives a Nak handshake.

16.6.14.1.2 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.

The USBERRINT bit in the USBSTS register is set and if the USB Error Interrupt Enable bit in the USBINTR register is set, then a hardware interrupt is signaled to the system at the next interrupt threshold. The host controller must never start an OUT transaction that babbles across a micro-frame EOF.

NOTE

When a host controller detects a data PID mismatch, it must either: disable the packet babble checking for the duration of the bus transaction or do packet babble checking based solely on Maximum Packet Size. The USB core specification defines the requirements on a data receiver when it receives a data PID mismatch (for example, expects a DATA0 and gets a DATA1 or visa-versa). In summary, it must ignore the received data and respond with an ACK handshake, in order to advance the transmitter's data sequence.The EHCI interface allows system software to provide buffers for a Control, Bulk or Interrupt IN endpoint that are not an even multiple of the maximum packet size specified by the device. Whenever a device misses an ACK for an IN endpoint, the host and device are out of synchronization with respect to the progress of the data transfer. The host controller may have advanced the transfer to a buffer that is less than maximum packet size. The device re-sends its maximum packet size data packet, with the original data PID, in response to the next IN token. In order to properly manage the bus protocol, the host controller must disable the packet babble check when it observes the data PID mismatch.

16.6.14.1.3 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*.

16.6.14.1.4 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 the USB Interrupt (USBINT) bit in the USBSTS register 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 the USB Interrupt Enable bit in the USBINTR register is set, a hardware interrupt is signaled to the system at the next interrupt threshold. If the completion is because of errors, the USBERRINT bit in the USBSTS register is also set.

16.6.14.1.5 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, the USBINT bit in the USBSTS register is set. If the USB Interrupt Enable bit is set in the USBINTR register, a hardware interrupt is signaled to the system at the next interrupt threshold.

16.6.14.2 Host Controller Event Interrupts

These interrupt sources are independent of the interrupt threshold (with the one exception being the Interrupt on Async Advance.

16.6.14.2.1 Port Change Events

Port registers contain status and status change bits. When the status change bits are set, the host controller sets the Port Change Detect bit in the USBSTS register. If the Port Change Interrupt Enable bit in the USBINTR register is set, then the host controller issues a hardware interrupt. The port status change bits include:

- Connect Status Change
- Port Enable/Disable Change
- Over-current Change
- Force Port Resume

16.6.14.2.2 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 in the USBSTS register. If the Frame List Rollover Enable bit in the USBINTR register is set, the host controller issues a hardware interrupt. This interrupt is not delayed to the next interrupt threshold.

16.6.14.2.3 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 the Interrupt on Async Advance Doorbell bit in the USBCMD register. If it is set, it sets the Interrupt on Async Advance bit in the USBSTS register. If the Interrupt on Async Advance Enable bit in the USBINTR register is set, the host controller issues a hardware interrupt at the next interrupt threshold. A detailed explanation of this feature is described in Section 16.6.9.2, “Removing Queue Heads from Asynchronous Schedule.”

16.6.14.2.4 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:

- The Run/Stop bit in the USBCMD register is cleared.
- The Host System Error and HCHalted bits in the USBSTS register are set:
- If the Host System Error Enable bit in the USBINTR register is set, then the host controller issues a hardware interrupt. This interrupt is not delayed to the next interrupt threshold.

Table 16-80 summarizes the required actions taken on the various host errors.

Table 16-80. 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 HCReset in the USBCMD register before re-initializing and restarting the host controller.

16.7 Device Data Structures—DR Module Only

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 core 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.

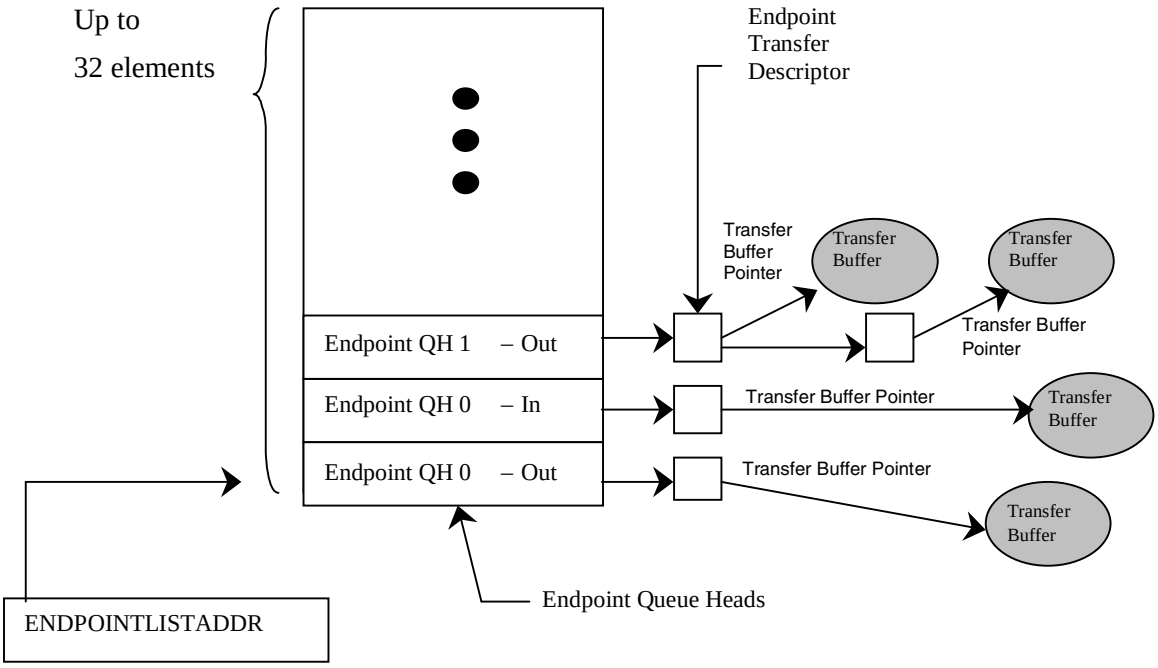


Figure 16-62. End Point Queue Head Organization

16.7.1 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 16-63 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[1]	0_0000	0x04
Next dTD Pointer[1]	0000 | T[1]	0x08[2]
00 | Total Bytes[1]	ioc[1] | 000 | MultO[1] | 00 | Status[1]	0x0C[2]
Buffer Pointer (Page 0)[1]	Current Offset[1]	0x10[2]
Buffer Pointer (Page 1)[1]	Reserved	0x14[2]
Buffer Pointer (Page 2)[1]	Reserved	0x18[2]
Buffer Pointer (Page 3)[1]	Reserved	0x1C[2]
Buffer Pointer (Page 4)[1]	Reserved	0x20[2]
Reserved		0x24
Set-up Buffer Bytes 3–0[1]		0x28
Set-up Buffer Bytes 7–4[1]		0x2C

Figure 16-63. Endpoint Queue Head Layout

1 Device controller read/write; all others read-only.

2 Offsets 0x08 through 0x20 contain the transfer overlay.

16.7.1.1 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 16-81. Endpoint Capabilities/Characteristics

Bit	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. Note: Non-ISO endpoints must set Mult="00". Note: ISO endpoints must set Mult="01", "10", or "11" as needed.
29	zlt	Zero length termination select. This bit is used to indicate when a zero length packet is used to terminate transfers where to total transfer length is a multiple. This bit is not relevant for Isochronous transfers. 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 16-81. Endpoint Capabilities/Characteristics (continued)

Bit	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.

16.7.1.2 Transfer Overlay

The seven DWords in the overlay area represent a transaction working space for the device controller. The general operational model is that the device controller can detect whether the overlay area contains a description of an active transfer. If it does not contain an active transfer, then it will not read the associated endpoint.

After an endpoint is readied, the dTD will be copied into this queue head overlay area by the device controller. Until a transfer is expired, software must not write the queue head overlay area or the associated transfer descriptor. When the transfer is complete, the device controller will write the results back to the original transfer descriptor and advance the queue.

See dTD for a description of the overlay fields.

16.7.1.3 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 16-82. Current dTD Pointer

Bit	Description
31–5	Current dtd. This field is a pointer to the dTD that is represented in the transfer overlay area. This field will be modified by the Device Controller to next dTD pointer during endpoint priming or queue advance.
4–0	Reserved, should be cleared. Bit reserved for future use and should be cleared.

16.7.1.4 Set-up Buffer

The set-up buffer is dedicated storage for the 8-byte data that follows a set-up PID.

NOTE

Each endpoint has a TX and an RX dQH associated with it, and only the RX queue head is used for receiving setup data packets.

Table 16-83. 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.

16.7.2 Endpoint Transfer Descriptor (dTD)

The dTD describes to the device controller the location and quantity of data to be sent/received for given transfer. The DCD should not attempt to modify any field in an active dTD except the Next Link Pointer, which should only be modified as described in section Managing Transfers with Transfer Descriptors.

31–5	4–1	0	offset
Next Link Pointer	0000	T	0x00

31–30	29–16	15	14–13	12–11	10–9	8–7	6–0	offset
00	Total Bytes[1]	ioc	000	MultO	00	Status[1]		0x04
Buffer Pointer (Page 0)				Current Offset[1]				0x08
Buffer Pointer (Page 1)				0	Frame Number[1]			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 16-64. Endpoint Transfer Descriptor (dTD)

[1] Device controller read/write; all others read-only.

Table 16-84. Next dTD Pointer

Bit	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 16-85. dTD Token

Bit	Description
31	Reserved, should be cleared. Bit reserved for future use and should be cleared.
30–16	Total Bytes. This field specifies the total number of bytes to be moved with this transfer descriptor. This field is decremented by the number of bytes actually moved during the transaction and only on the successful completion of the transaction. 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).** 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. It is not a requirement for IN transfers that Total Bytes To Transfer be an even multiple of Maximum Packet Length. If software builds such a transfer descriptor for an IN transfer, the last transaction will always be less that Maximum Packet Length.
15	Interrupt On Complete (IOC). This bit is used to indicate if USBINT is to be set in response to device controller being finished with this dTD.
14–12	Reserved, should be cleared. Bits reserved for future use and should be cleared.
11–10	Multiplier Override (MultO). This field can be used for transmit ISO's (that is, ISO-IN) to override the multiplier in the QH. This field must be zero for all packet types that are not transmit-ISO. Example: if QH.multiplier = 3; Maximum packet size = 8; Total Bytes = 15; MultiO = 0 [default] Three packets are sent: {Data2(8); Data1(7); Data0(0)} if QH.multiplier = 3; Maximum packet size = 8; Total Bytes = 15; MultiO = 2 Two packets are sent: {Data1(8); Data0(7)} For maximal efficiency, software should compute MultO = greatest integer of (Total Bytes / Max. Packet Size) except for the case when Total Bytes = 0; then MultO should be 1. **Note: Non-ISO and Non-TX endpoints must set MultO="00"**.
9–8	Reserved, should be cleared. Bits reserved for future use and should be cleared.
7–0	Status. This field is used by the Device Controller to communicate individual command execution states back to the Device Controller software. This field contains the status of the last transaction performed on this qTD. The bit encodings are: Bit Status Field Description 7 Active. 6 Halted. 5 Data Buffer Error. 3 Transaction Error. 4,2,0 Reserved.

Table 16-86. Buffer Pointer Page 0

Bit	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 16-87. Buffer Pointer Page 1

Bit	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 16-88. Buffer Pointer Pages 2–4

Bit	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

16.8 Device Operational Model—DR Module Only

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.

16.8.1 Device Controller Initialization

After hardware reset, the USB_DR is disabled until the Run/Stop bit is set to a '1'. In the disabled state, the pull-up on the USB D+ is not active which prevents an attach event from occurring. At a minimum, it is necessary to have the queue heads setup for endpoint zero before the device attach occurs. Shortly after the device is enabled, a USB reset will occur followed by setup packet arriving at endpoint 0. A Queue head must be prepared so that the device controller can store the incoming setup packet.

In order to initialize a device, the software should perform the following steps:

1. Set Controller Mode to device mode. Optionally set Streaming Disable in the USBMODE register.

NOTE

Transitioning from host mode to device mode requires a device controller reset before modifying USBMODE.

2. Optionally modify the BURSTSIZE register.

3. Program the PTS field of the PORTSC*n* register if using a non-ULPI PHY.
4. Set iodis_b bit in the CONTROL register to enable PHY interface.
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 - Device Data Structures.

6. Configure ENDPOINTLISTADDR Pointer.

   For additional information on ENDPOINTLISTADDR, refer to the register table.
7. Enable the microprocessor interrupt associated with the USB_DR and optionally change setting of ITC field in USBCMD register.

   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 Run/Stop bit to Run Mode.

   After the Run bit is set, a device reset will occur. The DCD must monitor the reset event and set the DEVICEADDR register, set the ENDPTCTRLx registers, and adjust the software state as described in the Bus Reset section of the following Port State and Control section below.

NOTE

Endpoint 0 is designed as a control endpoint only and does not need to be configured using ENDPTCTRL0 register.

It is also not necessary to initially prime Endpoint 0 because the first packet received will always be a setup packet. The contents of the first setup packet will require a response in accordance with USB device framework command set.

16.8.2 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 is set to a '1'. After receiving a reset on the bus, the port will enter the *defaultFS or defaultHS* state in accordance with the protocol reset described in Appendix C.2 of the USB Specification Rev. 2.0. The following state diagram depicts the state of a USB 2.0 device.

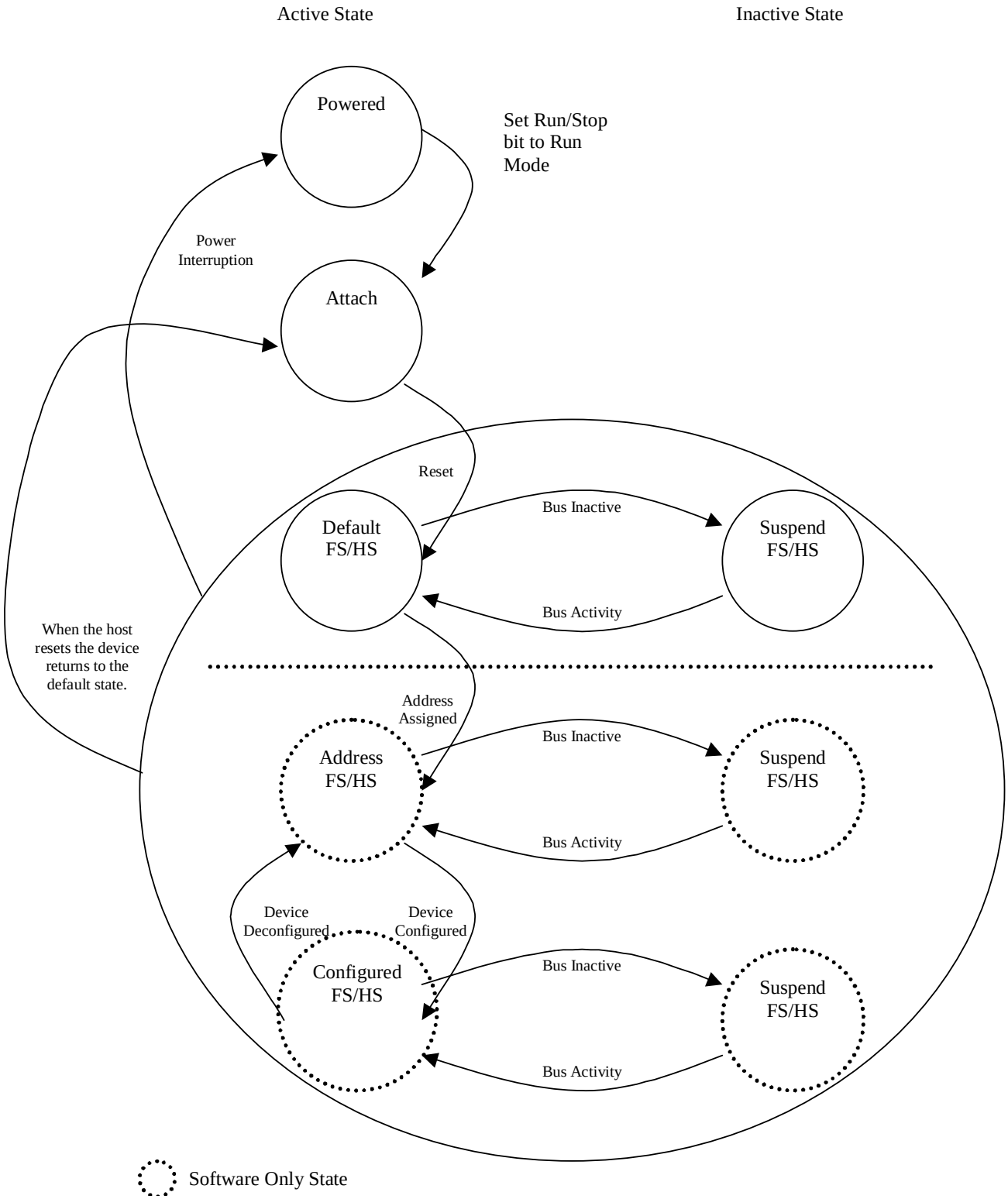


Figure 16-65. USB 2.0 Device States

States powered, attach, defaultFS/HS, suspendFS/HS are implemented in the USB_DR and are communicated to the DCD using the following status bits:

Table 16-89. Device Controller State Information Bits

Bit	Register
DCSuspend	USBSTS
USB Reset Received	USBSTS
Port Change Detect	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.

16.8.2.1 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 is set). After a reset is received, all endpoints (except endpoint 0) are disabled and any primed transactions will be cancelled by the device controller. The concept of priming will be clarified below, but the DCD must perform the following tasks when a reset is received:

Clear all setup token semaphores by reading the ENDPTSETUPSTAT register and writing the same value back to the ENDPTSETUPSTAT register.

Clear all the endpoint complete status bits by reading the ENDPTCOMPLETE register and writing the same value back to the ENDPTCOMPLETE register.

Cancel all primed status by waiting until all bits in the ENDPTPRIME are 0 and then writing 0xFFFF_FFFF to ENDPTFLUSH.

Read the reset bit in the PORTSC*n* register and make sure that it is still active. A USB reset will occur for a minimum of 3 ms and the DCD must reach this point in the reset cleanup before end of the reset occurs, otherwise a hardware reset of the device controller is recommended (rare.)

- A hardware reset can be performed by writing a one to the USB_DR reset bit in the USBCMD reset. Note: a hardware reset will cause the device to detach from the bus by clearing the Run/Stop bit. Thus, the DCD must completely re-initialize the USB_DR after a hardware reset.

Free all allocated dTDs because they will no longer be executed by the device controller. If this is the first time the DCD is processing a USB reset event, then it is likely that no dTDs have been allocated.

At this time, the DCD may release control back to the OS because no further changes to the device controller are permitted until a Port Change Detect is indicated.

After a Port Change Detect, the device has reached the default state and the DCD can read the PORTSC*n* 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.

16.8.2.2 Suspend/Resume

16.8.2.2.1 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 by 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.

16.8.2.2.2 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 DCSuspend bit in the PORTSC*n* is set to a '1', 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.

16.8.2.2.3 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 Resume bit in the in the PORTSC*n* while the device is in suspend state. Sending resume signal to an upstream port

should cause the host to issue resume signaling and bring the suspended bus segment (one more devices) back to the active condition.

NOTE

Before resume signaling can be used, the host must enable it by using the Set Feature command defined in device framework (chapter 9) of the USB 2.0 Specification.

16.8.3 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 six(6) 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.

16.8.3.1 Endpoint Initialization

After hardware reset, all endpoints except endpoint zero are uninitialized and disabled. The DCD must configure and enable each endpoint by writing to configuration bit in the ENDPTCTRL*n* register. Each 32-bit ENDPTCTRL*n* is split into an upper and lower half. The lower half of ENDPTCTRL*n* is used to configure the receive or OUT endpoint and the upper half is likewise used to configure the corresponding transmit or IN endpoint. Control endpoints must be configured the same in both the upper and lower half of the ENDPTCTRL*n* register otherwise the behavior is undefined. The following table shows how to construct a configuration word for endpoint initialization.

Table 16-90. 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'

16.8.3.1.1 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 ENDPTCTRL*n* register associated with the given endpoint and the given direction. In a functional stall condition, the device controller will continue to return STALL responses to all transactions occurring on the respective endpoint and direction until the endpoint stall bit is cleared by the DCD.

A protocol stall, unlike a function stall, is used on control endpoints is automatically cleared by the device controller at the start of a new control transaction (setup phase). When enabling a protocol stall, the DCD should enable the stall bits (both directions) as a pair. A single write to the ENDPTCTRL*n* register can ensure that both stall bits are set at the same instant.

NOTE

Any write to the ENDPTCTRL*n* register during operational mode must preserve the endpoint type field (that is, perform a read-modify-write).

Table 16-91. 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

16.8.3.2 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 USB 2.0 specification.

16.8.3.2.1 Data Toggle Reset

The DCD may reset the data toggle state bit and cause the data toggle sequence to reset in the device controller by writing a '1' to the data toggle reset bit in the ENDPTCTRL*n* register. This should only be necessary when configuring/initializing an endpoint or returning from a STALL condition.

16.8.3.2.2 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.

16.8.3.3 Device Operational Model For Packet Transfers

All transactions on the USB bus are initiated by the host and in turn, the device must respond to any request from the host within the turnaround time stated in the USB 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 3 (transmit direction) is configured as a bulk pipe, then we can 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 will be 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.

16.8.3.3.1 Priming Transmit Endpoints

Priming a transmit endpoint will cause the device controller to fetch the device transfer descriptor (dTD) for the transaction pointed to by the device queue head (dQH). After the dTD is fetched, it will be stored in the dQH until the device controller completes the transfer described by the dTD. Storing the dTD in the dQH allows the device controller to fetch the operating context needed to handle a request from the host without the need to follow the linked list, starting at the dQH when the host request is received.

After the device has loaded the dTD, the leading data in the packet is stored in a FIFO in the device controller. This FIFO is split into virtual channels so that the leading data can be stored for any endpoint up to the maximum number of endpoints configured at device synthesis time.

After a priming request is complete, an endpoint state of primed is indicated in the ENDPTSTATUS register. For a primed transmit endpoint, the device controller can respond to an IN request from the host and meet the stringent bus turnaround time of High Speed USB.

Since only the leading data is stored in the device controller FIFO, it is necessary for the device controller to begin filling in behind leading data after the transaction starts. The FIFO must be sized to account for the maximum latency that can be incurred by the system memory bus.

16.8.3.3.2 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.

16.8.3.4 Interrupt/Bulk Endpoint Operational Model

The behaviors of the device controller for interrupt and bulk endpoints are identical. All valid IN and OUT transactions to bulk pipes will handshake with a NAK unless the endpoint had been primed. Once the endpoint has been primed, data delivery will commence.

A dTD will be retired by the device controller when the packets described in the transfer descriptor have been completed. Each dTD describes N packets to be transferred according to the USB Variable Length transfer protocol. The formula and table on the following page describe how the device controller computes the number and length of the packets to be sent/received by the USB vary according to the total number of bytes and maximum packet length.

With Zero Length Termination (ZLT) = 0

N = INT(Number Of Bytes/Max. Packet Length) + 1

With Zero Length Termination (ZLT) = 1

N = MAXINT(Number Of Bytes/Max. Packet Length)

Table 16-92. 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 16-93. Variable Length Transfer Protocol Example (ZLT=1)

Bytes (dTD)	Max. Packet Length (dQH)	N	P1	P2	P3
511	256	2	256	255	
512	256	2	256	256	
512	512	1	512		

NOTE

The MULT field in the dQH must be set to ‘00’ for bulk, interrupt, and control endpoints.

TX-dTD is complete when:

- All packets described dTD were successfully transmitted. *** Total bytes in dTD will equal zero when this occurs.

RX-dTD is complete when:

- All packets described in dTD were successfully received. *** Total bytes in dTD will equal zero when this occurs.
- A short packet (number of bytes < maximum packet length) was received. *** This is a successful transfer completion; DCD must check Total Bytes in dTD to determine the number of bytes that are remaining. From the total bytes remaining in the dTD, the DCD can compute the actual bytes received.
- A long packet was received (number of bytes > maximum packet size) OR (total bytes received > total bytes specified). *** This is an error condition. The device controller will discard the remaining packet, and set the Buffer Error bit in the dTD. In addition, the endpoint will be flushed and the USBERR interrupt will become active.

On the successful completion of the packet(s) described by the dTD, the active bit in the dTD will be cleared and the next pointer will be followed when the Terminate bit is clear. When the Terminate bit is set, the USB_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 will be left pointing to the dTD that was in error. In order to recover from this error condition, the DCD must properly re-initialize the dQH by clearing the active bit and update the nextTD pointer before attempting to re-prime the endpoint.

NOTE

All packet level errors such as a missing handshake or CRC error will be retried automatically by the device controller.

There is no required interaction with the DCD for handling such errors.

16.8.3.4.1 Interrupt/Bulk Endpoint Bus Response Matrix

Table 16-94. Interrupt/Bulk Endpoint Bus Response Matrix

	Stall	Not Primed	Primed	Underflow	Overflow
Setup	Ignore	Ignore	Ignore	N/A	N/A
In	STALL	NAK	Transmit	BS Error[1]	N/A
Out	STALL	NAK	Receive + NYET/ACK[2]	N/A	NAK
Ping	STALL	NAK	ACK	N/A	N/A
Invalid	Ignore	Ignore	Ignore	Ignore	Ignore

1 Force Bit Stuff Error

2 NYET/ACK - NYET unless the Transfer Descriptor has packets remaining according to the USB variable length protocol then ACK.

SYSERR - System error should never occur when the latency FIFOs are correctly sized and the DCD is responsive.

16.8.3.5 Control Endpoint Operation Model

16.8.3.5.1 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 be an incoming setup packet. This is the preferred behavior because ignoring repeated setup packets due to long software interrupt latency would be a compliance issue.

Setup Packet Handling

- Disable Setup Lockout by writing '1' to Setup Lockout Mode (SLOM) in USBMODE. (once at initialization). Setup lockout is not necessary when using the tripwire as described below.

NOTE

Leaving the Setup Lockout Mode As '0' will result in a potential compliance issue.

- After receiving an interrupt and inspecting ENDPTSETUPSTAT to determine that a setup packet was received on a particular pipe:
  - Write '1' to clear corresponding bit ENDPTSETUPSTAT.
  - Write '1' to Setup Tripwire (SUTW) in USBCMD register.
  - Duplicate contents of dQH.SetupBuffer into local software byte array.

— Read Setup TripWire (SUTW) in USBCMD register. (if set - continue; if cleared - goto 2)
— Write '0' to clear Setup Tripwire (SUTW) in USBCMD register.
— Process setup packet using local software byte array copy and execute status/handshake phases.

Note: After receiving a new setup packet the status and/or handshake phases may still be pending from a previous control sequence. These should be flushed and de-allocated before linking a new status and/or handshake dTD for the most recent setup packet.

16.8.3.5.2 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.

16.8.3.5.3 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.

16.8.3.5.4 Control Endpoint Bus Response Matrix

Shown in the following table is the device controller response to packets on a control endpoint according to the device controller state.

Table 16-95. Control Endpoint Bus Response Matrix

Token Type	Endpoint State					Setup Lockout
	Stall	Not Primed	Primed	Underflow	Overflow	
Setup	ACK	ACK	ACK	N/A	SYSERR[1]	
In	STALL	NAK	Transmit	BS Error[2]	N/A	N/A
Out	STALL	NAK	Receive + NYET/ACK[3]	N/A	NAK	N/A
Ping	STALL	NAK	ACK	N/A	N/A	N/A
Invalid	Ignore	Ignore	Ignore	Ignore	Ignore	Ignore

1 SYSERR - System error should never occur when the latency FIFOs are correctly sized and the DCD is responsive.

2 Force Bit Stuff Error

3 NYET/ACK - NYET unless the Transfer Descriptor has packets remaining according to the USB variable length protocol then ACK.

16.8.3.6 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 is accomplished by the following:

- Exactly MULT Packets per (micro)Frame are transmitted/received. Note: MULT is a two-bit field in the device Queue Head. The variable length packet protocol is not used on isochronous endpoints.
- NAK responses are not used. Instead, zero length packets and sent in response to an IN request to an unprimed endpoints. For unprimed RX endpoints, the response to an OUT transaction is to ignore the packet within the device controller.
- Prime requests always schedule the transfer described in the dTD for the next (micro)frame. If the ISO-dTD is still active after that frame, then the ISO-dTD will be held ready until executed or canceled by the DCD.

The USB_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 will be cleared as usual to indicate to software that the device controller completed a priming the dTD for transfer. Internal to the design, the device controller hardware masks that

prime start until the next frame boundary. This behavior is hidden from the DCD but occurs so that the device controller can match the dTD to a specific (micro)frame.

Another difference with isochronous endpoints is that the transaction must wholly complete in a (micro)frame. Once an ISO transaction is started in a (micro)frame it will retire the corresponding dTD when MULT transactions occur or the device controller finds a fulfillment condition.

The transaction error bit set in the status field indicates a fulfillment error condition. When a fulfillment error occurs, the frame after the transfer failed to complete wholly, the device controller will force retire the ISO-dTD and move to the next ISO-dTD.

It is important to note that fulfillment errors are only caused due to partially completed packets. If no activity occurs to a primed ISO-dTD, the transaction will stay primed indefinitely. This means it is up to software discard transmit ISO-dTDs that pile up from a failure of the host to move the data.

Finally, the last difference with ISO packets is in the data level error handling. When a CRC error occurs on a received packet, the packet is not retried similar to bulk and control endpoints. Instead, the CRC is noted by setting the Transaction Error bit and the data is stored as usual for the application software to sort out.

- TX Packet Retired
  - MULT counter reaches zero.
  - Fulfillment Error [Transaction Error bit is set]
  - #Packets Occurred > 0 AND # Packets Occurred < MULT

NOTE

For TX-ISO, MULT Counter can be loaded with a lesser value in the dTD Multiplier Override field. If the Multiplier Override is zero, the MULT Counter is initialized to the Multiplier in the QH.

- RX Packet Retired:
  - MULT counter reaches zero.
  - Non-MDATA Data PID is received
  - Overflow Error:
  - Packet received is > maximum packet length. [Buffer Error bit is set]
  - Packet received exceeds total bytes allocated in dTD. [Buffer Error bit is set]
  - Fulfillment Error [Transaction Error bit is set]
  - # Packets Occurred > 0 AND # Packets Occurred < MULT
  - CRC Error [Transaction Error bit is set]

NOTE

For ISO, when a dTD is retired, the next dTD is primed for the next frame. For continuous (micro)frame to (micro)frame operation the DCD should ensure that the dTD linked-list is out ahead of the device controller by at least two (micro)frames.

16.8.3.6.1 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.

16.8.3.6.2 Isochronous Endpoint Bus Response Matrix

Table 16-96. Isochronous Endpoint Bus Response Matrix

	Stall	Not Primed	Primed	Underflow	Overflow
Setup	STALL	STALL	STALL	N/A	N/A
In	NULL[1] Packet	NULL Packet	Transmit	BS Error[2]	N/A
Out	Ignore	Ignore	Receive	N/A	Drop Packet
Ping	Ignore	Ignore	Ignore	Ignore	Ignore
Invalid	Ignore	Ignore	Ignore	Ignore	Ignore

1 Zero Length Packet

2 Force Bit Stuff Error

16.8.4 Managing Queue Heads

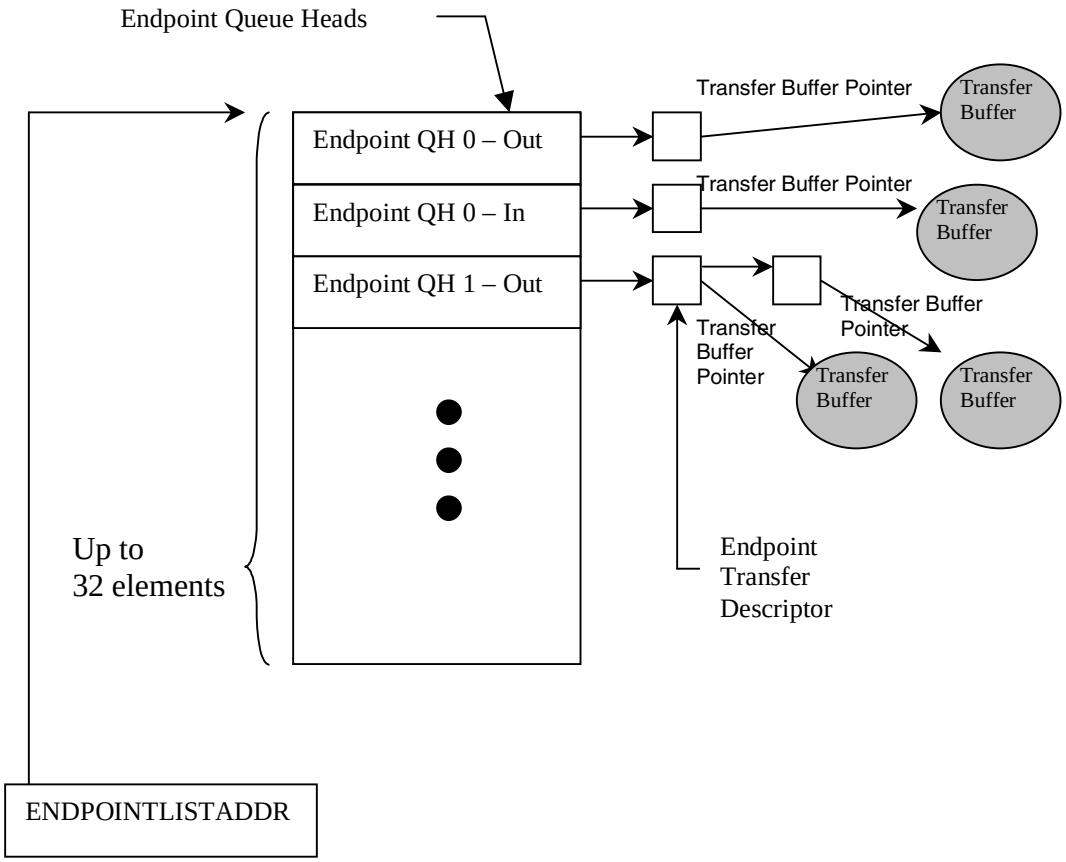


Figure 16-66. Endpoint Queue Head Diagram

The device queue head (dQH) points to the linked list of transfer tasks, each depicted by the device Transfer Descriptor (dTD). An area of memory pointed to by ENDPOINTLISTADDR contains a group of all dQH's in a sequential list as shown in Figure 79. The even elements in the list of dQH's are used for receive endpoints (OUT/SETUP) and the odd elements are used for transmit endpoints (IN/INTERRUPT). Device transfer descriptors are linked head to tail starting at the queue head and ending at a terminate bit. Once the dTD has been retired, it will no longer be part of the linked list from the queue head. Therefore, software is required to track all transfer descriptors since pointers will no longer exist within the queue head once the dTD is retired (see section Software Link Pointers).

In addition to the current and next pointers and the dTD overlay examined in section Operational Model For Packet Transfers, the dQH also contains the following parameters for the associated endpoint: Multipler, Maximum Packet Length, Interrupt On Setup. The complete initialization of the dQH including these fields is demonstrated in the next section.

16.8.4.1 Queue Head Initialization

One pair of device queue heads must be initialized for each active endpoint. To initialize a device queue head:

- Write the wMaxPacketSize field as required by the USB Chapter 9 or application specific protocol.
- Write the multiplier field to 0 for control, bulk, and interrupt endpoints. For ISO endpoints, set the multiplier to 1,2, or 3 as required bandwidth an in conjunction with the USB Chapter 9 protocol. Note: In FS mode, the multiplier field can only be 1 for ISO endpoints.
- Write the next dTD Terminate bit field to ‘1.’
- Write the Active bit in the status field to ‘0.’
- Write the Halt bit in the status field to ‘0.’

NOTE

The DCD must only modify dQH if the associated endpoint is not primed and there are no outstanding dTD's.

16.8.4.2 Operational Model For Setup Transfers

As discussed in Section 16.8.3.5, “Control Endpoint Operation Model,” setup transfer requires special treatment by the DCD. A setup transfer does not use a dTD but instead stores the incoming data from a setup packet in an 8-byte buffer within the dQH.

Upon receiving notification of the setup packet, the DCD should handle the setup transfer as demonstrated here:

1. Copy setup buffer contents from dQH - RX to software buffer.
2. Acknowledge setup backup by writing a ‘1’ to the corresponding bit in ENDPTSETUPSTAT.

NOTE

The acknowledge must occur before continuing to process the setup packet.

NOTE

After the acknowledge has occurred, the DCD must not attempt to access the setup buffer in the dQH - RX. Only the local software copy should be examined.

3. Check for pending data or status dTD's from previous control transfers and flush if any exist as discussed in section Flushing/De-priming an Endpoint.

NOTE

It is possible for the device controller to receive setup packets before previous control transfers complete. Existing control packets in progress must be flushed and the new control packet completed.

4. Decode setup packet and prepare data phase [optional] and status phase transfer as required by the USB Chapter 9 or application specific protocol.

16.8.5 Managing Transfers with Transfer Descriptors

16.8.5.1 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.

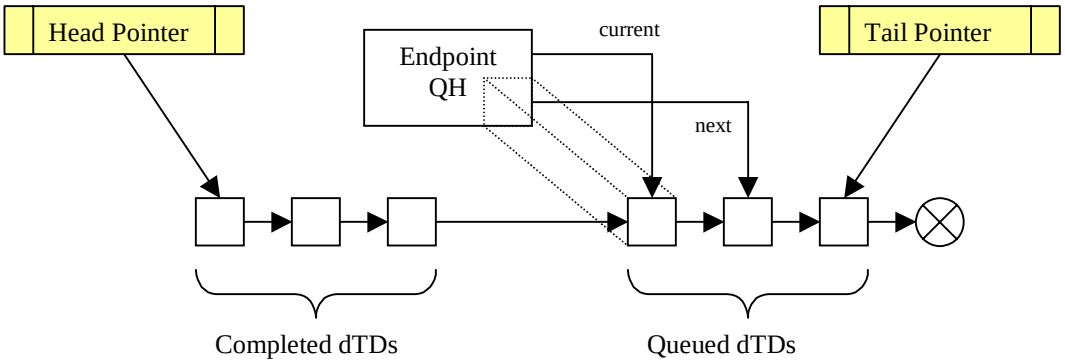


Figure 16-67. Software Link Pointers

16.8.5.2 Building a Transfer Descriptor

Before a transfer can be executed from the linked list, a dTD must be built to describe the transfer. Use the following procedure for building dTDs.

Allocate 8-DWord dTD block of memory aligned to 8-DWord boundaries. Example: bit address 4–0 would be equal to ‘00000’

Write the following fields:

1. Initialize first 7 DWords to 0.
2. Set the terminate bit to ‘1.’
3. Fill in total bytes with transfer size.
4. Set the interrupt on complete if desired.
5. Initialize the status field with the active bit set to ‘1’ and all remaining status bits set to ‘0.’
6. Fill in buffer pointer page 0 and the current offset to point to the start of the data buffer.
7. Initialize buffer pointer page 1 through page 4 to be one greater than each of the previous buffer pointer.

16.8.5.3 Executing A Transfer Descriptor

To safely add a dTD, the DCD must be follow this procedure which will handle the event where the device controller reaches the end of the dTD list at the same time a new dTD is being added to the end of the list.

Determine whether the link list is empty:

Check DCD driver to see if pipe is empty (internal representation of linked-list should indicate if any packets are outstanding)

Case 1: Link list is empty

1. Write dQH next pointer AND dQH terminate bit to 0 as a single DWord operation.
2. Clear active and halt bit in dQH (in case set from a previous error).
3. Prime endpoint by writing '1' to correct bit position in ENDPTPRIME.

Case 2: Link list is not empty

1. Add dTD to end of linked list.
2. Read correct prime bit in ENDPTPRIME - if '1' DONE.
3. Set ATDTW bit in USBCMD register to '1'.
4. Read correct status bit in ENDPTSTATUS. (store in tmp. variable for later)
5. Read ATDTW bit in USBCMD register.

   If '0' goto 3.

   If '1' continue to 6.
6. Write ATDTW bit in USBCMD register to '0'.
7. If status bit read in (3) is '1' DONE.
8. If status bit read in (3) is '0' then Goto Case 1: Step 1.

16.8.5.4 Transfer Completion

After a dTD has been initialized and the associated endpoint primed the device controller will execute the transfer upon the host-initiated request. The DCD will be notified with a USB interrupt if the Interrupt On Complete bit was set or alternately, the DCD can poll the endpoint complete register to find when the dTD had been executed. After a dTD has been executed, DCD can check the status bits to determine success or failure.

CAUTION

Multiple dTD can be completed in a single endpoint complete notification. After clearing the notification, DCD must search the dTD linked list and retire all dTDs that have finished (Active bit cleared).

By reading the status fields of the completed dTDs, the DCD can determine if the transfers completed successfully. Success is determined with the following combination of status bits:

- Active = 0
- Halted = 0

- Transaction Error = 0
- Data Buffer Error = 0

Should any combination other than the one shown above exist, the DCD must take proper action. Transfer failure mechanisms are indicated in the Device Error Matrix.

In addition to checking the status bit the DCD must read the Transfer Bytes field to determine the actual bytes transferred. When a transfer is complete, the Total Bytes transferred is by decremented by the actual bytes transferred. For Transmit packets, a packet is only complete after the actual bytes reaches zero, but for receive packets, the host may send fewer bytes in the transfer according the USB variable length packet protocol.

16.8.5.5 Flushing/De-priming an Endpoint

It is necessary for the DCD to flush to de-prime one more endpoints on a USB device reset or during a broken control transfer. There may also be application specific requirements to stop transfers in progress. The following procedure can be used by the DCD to stop a transfer in progress:

1. Write a '1' to the corresponding bit(s) in ENDPTFLUSH.
2. Wait until all bits in ENDPTFLUSH are '0'.
3. Software note: this operation may take a large amount of time depending on the USB bus activity. It is not desirable to have this wait loop within an interrupt service routine.
4. Read ENDPTSTATUS to ensure that for all endpoints commanded to be flushed, that the corresponding bits are now '0'. If the corresponding bits are '1' after step #2 has finished, then the flush failed as described in the following:

   Explanation: In very rare cases, a packet is in progress to the particular endpoint when commanded flush using ENDPTFLUSH. A safeguard is in place to refuse the flush to ensure that the packet in progress completes successfully. The DCD may need to repeatedly flush any endpoints that fail to flush be repeating steps 1-3 until each endpoint is successfully flushed.

16.8.5.6 Device Error Matrix

The following table summarizes packet errors that are not automatically handled by the USB_DR

Table 16-97. 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 16-98. Error Descriptions

Overflow	Number of bytes received exceeded max. packet size or total buffer length. ** This error will also set the Halt bit in the dQH and if there are dTDs remaining in the linked list for the endpoint, then those will not be executed.
ISO Packet Error	CRC Error on received ISO packet. Contents not guaranteed to be correct.
ISO Fulfillment Error	Host failed to complete the number of packets defined in the dQH mult field within the given (micro)frame. For scheduled data delivery the DCD may need to readjust the data queue because a fulfillment error will cause Device Controller to cease data transfers on the pipe for one (micro)frame. During the 'dead'(micro)frame, the Device Controller reports error on the pipe and primes for the following frame.

16.8.6 Servicing Interrupts

The interrupt service routine must consider that there are high-frequency, low-frequency operations, and error operations and order accordingly.

16.8.6.1 High-Frequency Interrupts

High frequency interrupts in particular should be handed in the order below. The most important of these is listed first because the DCD must acknowledge a setup buffer in the timeliest manner possible.

Table 16-99. Interrupt Handling Order

Execution Order	Interrupt	Action
1a	USB Interrupt[1] ENDPTSETUPSTATUS	Copy contents of setup buffer and acknowledge setup packet (as indicated in section Managing Queue Heads). Process setup packet according to USB 2.0 Chapter 9 or application specific protocol.
1b	USB Interrupt ENDPTCOMPLETE	Handle completion of dTD as indicated in section Managing Queue Heads.
2	SOF Interrupt	Action as deemed necessary by application. This interrupt may not have a use in all applications.

[1] It is likely that multiple interrupts to stack up on any call to the Interrupt Service Routine AND during the Interrupt Service Routine.

16.8.6.2 Low-Frequency Interrupts

The low frequency events include the following interrupts. These interrupt can be handled in any order since they don't occur often in comparison to the high-frequency interrupts.

Table 16-100. 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.

16.8.6.3 Error Interrupts

Error interrupts will be least frequent and should be placed last in the interrupt service routine.

Table 16-101. 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.

16.9 Deviations from the EHCI Specifications

The host mode operation of the modules is nearly EHCI-compatible with few minor differences. For the most part, the modules conform 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 (MPH module only)—Allows direct attachment of FS and LS devices in host mode without the need for a companion controller.
- Device operation (DR module only)—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 modules do 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.

16.9.1 Embedded Transaction Translator Function—MPH Module Only

The MPH 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.

16.9.1.1 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 16.3.1.3, “Host Controller Structural Parameters (HCSPARAMS)” for usage information.

16.9.1.2 Operational Registers

The following additions have been added to the operational registers to support the embedded TT:

- ASYNCTTSTS is a new register on the MPH module.
- Addition of two-bit Port Speed (PSPD) to the PORTSCn register.

16.9.1.3 Discovery

In a standard EHCI controller design, the EHCI host controller driver detects a Full speed (FS) or Low speed (LS) device by noting if the port enable bit is set after the port reset operation. The port enable will only be set in a standard EHCI controller implementation after the port reset operation and when the host and device negotiate a High-Speed connection (that is, Chirp completes successfully).

The module will always set the port enable after the port reset operation regardless of the result of the host device chirp result and the resulting port speed will be indicated by the PSPD field in PORTSC*n*. Therefore, the standard EHCI host controller driver requires an alteration to handle directly connected Full and Low speed devices or hubs. The change is a fundamental one in that is summarized in Table 16-102.

Table 16-102. 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*n*.
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*n*.
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)]

16.9.1.4 Data Structures

The same data structures used for FS/LS transactions though a HS hub are also used for transactions through the Root Hub. Here it is demonstrated how the Hub Address and Endpoint Speed fields should be set for directly attached FS/LS devices and hubs:

1. QH (for direct attach FS/LS) – Async. (Bulk/Control Endpoints) Periodic (Interrupt)

- Hub Address = 0
- Transactions to direct attached device/hub.
  - — QH.EPS = Port Speed
- Transactions to a device downstream from direct attached FS hub.
  - — QH.EPS = Downstream Device Speed

NOTE

When QH.EPS = 01 (LS) and PORTSC*n*[PSPD] = 00 (FS), a LS-pre-pid will be sent before the transmitting LS traffic.

Maximum Packet Size must be less than or equal 64 or undefined behavior may result.

2. siTD (for direct attach FS) – Periodic (ISO Endpoint)

- All FS ISO transactions:
  - — Hub Address = 0
  - — siTD.EPS = 00 (full speed)

  Maximum Packet Size must less than or equal to 1023 or undefined behavior may result.

16.9.1.5 Operational Model

The operational models are well defined for the behavior of the Transaction Translator (see USB 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 MPH 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.

16.9.1.5.1 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 USB 2.0 specification for a Hub-based Transaction Translator.

All periodic transfers always begin at B-frame 0 (after SOF) and continue until the stored periodic transfers are complete. As an example of the microframe pipeline implemented in the embedded Transaction Translator, all periodic transfers that are tagged in EHCI to execute in H-frame 0 will be ready to execute on the bus in B-frame 0.

It is important to note that when programming the S-mask and C-masks in the EHCI data structures to schedule periodic transfers for the embedded Transaction Translator, the EHCI host controller driver must

follow the same rules specified in EHCI for programming the S-mask and C-mask for downstream Hub-based Transaction Translators.

Once periodic transfers are exhausted, any stored asynchronous transfer will be moved. Asynchronous transfers are opportunistic in that they shall execute whenever possible and their operation is not tied to H-frame and B-frame boundaries with the exception that an asynchronous transfer can not babble through the SOF (start of B-frame 0.)

16.9.1.5.2 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 16-103 summarizes the conditions where handshakes are emulated from internal state instead of actual handshakes to HS split bus traffic.

Table 16-103. Emulated Handshakes

Condition	Emulate TT Response
Start-Split: All asynchronous buffers full.	NAK
Start-Split: All periodic buffers full.	ERR
Start-Split: Success for start of Async. Transaction.	ACK
Start-Split: Start Periodic Transaction.	No Handshake (Ok)
Complete-Split: Failed to find transaction in queue.	Bus Time Out
Complete-Split: Transaction in Queue is Busy.	NYET
Complete-Split: Transaction in Queue is Complete.	[Actual Handshake from FS/LS device]

16.9.1.5.3 Asynchronous Transaction Scheduling and Buffer Management

The following USB 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 though the use of the Error! Reference source not found. register.

16.9.1.5.4 Periodic Transaction Scheduling and Buffer Management

The following USB 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
- USB 2.0 - 11.18.[7-8]
  - — Transaction tracking for up to 16 data pipes.
    - – Some applications may not require transaction tracking up to a maximum of 16 periodic data pipes. The option to limit the tracking to only 4 periodic data pipes exists in the by changing the configuration constant VUSB_HS_TT_PERIODIC_CONTEXTS to 4. The result is a significant gate count savings to the core given the limitations implied.

WARNING

Limiting the number of tracking pipes in the EMBedded - TT to four (4) will impose the restriction that no more than 4 periodic transactions (INTERRUPT/ISOCHRONOUS) can be scheduled through the embedded-tt per frame. The number 16 was chosen in the USB specification because it is sufficient to ensure that the high-speed to full-speed periodic pipeline can remain full. Keeping the pipeline full puts no constraint on the number of periodic transactions that can be scheduled in a frame and the only limit becomes the flight time of the packets on the bus.

  - — Complete-split transaction searching.

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.

16.9.1.5.5 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 16.3.1.3, “Host Controller Structural Parameters (HCSPARAMS),” for more information.

16.9.2 Device Operation—DR Module Only

The co-existence of a device operational controller within the DR module has little effect on EHCI compatibility for host operation. However, given that the DR controller is initialized in neither host nor device mode, the USBMODE register must be programmed for host operation before the EHCI host controller driver can begin EHCI host operations.

16.9.3 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).

16.9.4 SOF Interrupt

The SOF interrupt is a free running 125 µsec interrupt for host mode. EHCI does not specify this interrupt, but it has been added for convenience and as a potential software time base. Note that the free running interrupt is shared with the DR-only device-mode start-of-frame interrupt. See Section 16.3.2.2, “USB Status Register (USBSTS),” and Section 16.3.2.3, “USB Interrupt Enable Register (USBINTR),” for more information.

16.9.5 Embedded Design

This is an Embedded USB Host Controller as defined by the EHCI specification and thus does not implement the PCI configuration registers.

16.9.5.1 Frame Adjust Register

Given that the optional PCI configuration registers are not included in this implementation, there is no corresponding bit level timing adjustments like those provided by the Frame Adjust register in the PCI configuration registers. Starts of microframes are timed precisely to 125 µsec using the transceiver clock as a reference clock. That is, 60 Mhz transceiver clock for 8-bit physical interfaces and full-speed serial interfaces or 30 Mhz transceiver clock for 16-bit physical interfaces.

16.9.6 Miscellaneous Variations from EHCI

16.9.6.1 Programmable Physical Interface Behavior

The modules support multiple physical interfaces which can operate in different modes when the module is configured with the software programmable Physical Interface Modes. The control bits for selecting the

PHY operating mode have been added to the PORTSC*n* register providing a capability that is not defined by the EHCI specification.

16.9.6.2 Discovery

16.9.6.2.1 Port Reset

The port connect methods specified by EHCI require setting the port reset bit in the register for a duration of 10 msec. Due to the complexity required to support the attachment of devices that are not high speed there are counter already present in the design that can count the 10 msec reset pulse to alleviate the requirement of the software to measure this duration. Therefore, the basic connection is then summarized as the following:

- [Port Change Interrupt] Port connect change occurs to notify the host controller driver that a device has attached.
- Software shall write a ‘1’ to the reset the device.
- *Software shall write a ‘0’ to the reset the device after 10 msec.*
  - This step, which is necessary in a standard EHCI design, may be omitted with this implementation. Should the EHCI host controller driver attempt to write a ‘0’ to the reset bit while a reset is in progress the write will simple be ignored and the reset will continue until completion.
- [Port Change Interrupt] Port enable change occurs to notify the host controller that the device in now operational and at this point the port speed has been determined.

16.9.6.2.2 Port Speed Detection

After the port change interrupt indicates that a port is enabled, the EHCI stack should determine the port speed. Unlike the EHCI implementation which will re-assign the port owner for any device that does not connect at High-Speed, this host controller supports direct attach of non-HS devices. Therefore, the following differences are important regarding port speed detection:

- Port Owner is read-only and always reads 0.
- A 2-bit Port Speed indicator has been added to PORTSC to provide the current operating speed of the port to the host controller driver.
- A 1-bit High Speed indicator has been added to PORTSC to signify that the port is in HS vs. FS/LS – *This information is redundant with the 2-bit Port Speed indicator above*.

Chapter 17
I²C Interfaces

This chapter describes the two inter-IC (IIC or I²C) bus interfaces implemented on this device. Note that for most intents, the I²C interfaces are identical and are described as a single generic controller. Where necessary, differences between the two controllers are noted.

17.1 Introduction

The inter-IC (IIC or I²C) 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 the I²C interface.

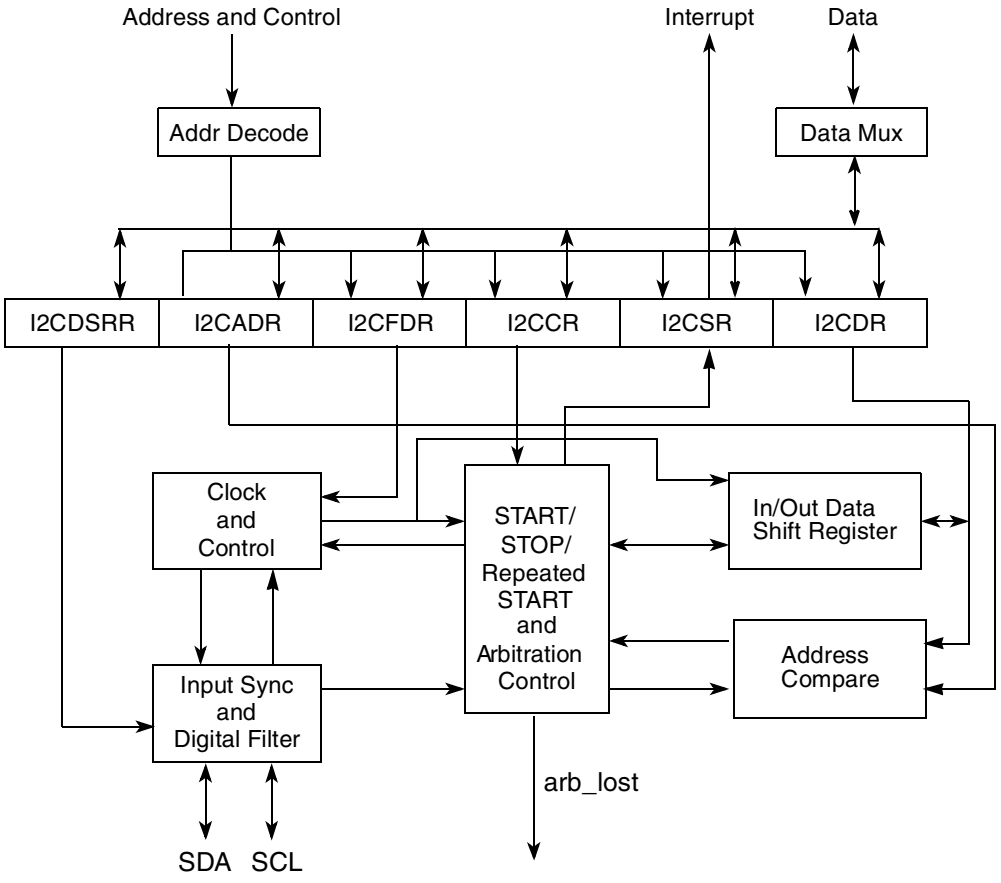


Figure 17-1. I²C Block Diagram

The two-wire I²C bus minimizes interconnections between devices. The synchronous, multiple-master I²C 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.

17.1.1 Features

Each I²C 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 I²C unit on this device can operate in one of the following modes:

- Master mode. The I²C initiates a transfer, generates clock signals, and terminates a transfer. It cannot use its own slave address as a calling address. The I²C cannot be a master and a slave simultaneously.
- Slave mode. The I²C is addressed by an I²C master. The module must be enabled before a START condition from an I²C master is detected.
- Interrupt-driven byte-to-byte data transfer. When successful slave addressing is achieved (and SCL*n* 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. I²C1 controller supports boot sequencer mode. This mode can be used to initialize the configuration registers in the device after the I²C 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. This mode is not supported by I²C2 controller.
- Reset configuration load (I²C1 only). In this mode, the I²C1 interface 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, I²C1 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 4.3.3.2, “Loading from I2C EEPROM.”

Additionally, the following three I²C–specific states are defined for the I²C 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 I²C interface uses the SDA*n* and SCL*n* signals, described in Table 17-1, for data transfer. Note that the signal patterns driven on SDA*n* represent address, data, or read/write information at different stages of the protocol.

Table 17-1. I²C Interface Signal Description

Signal Name	Idle State	I/O	State Meaning
Serial Clock (SCL1, SCL2)	High	I	When the I²C module is idle or acts as a slave, SCL*n* defaults as an input. The unit uses SCL*n* to synchronize incoming data on SDA*n*. The bus is assumed to be busy when SCL*n* is detected low.
		O	As a master, the I²C module drives SCL*n* along with SDA*n* when transmitting. As a slave, the I²C module drives SCL*n* negates for data pacing.
Serial Data (SDA1, SDA2)	High	I	When the I²C module is idle or in a receiving mode, SDA*n* defaults as an input. The unit receives data from other I²C devices on SDA*n*. The bus is assumed to be busy when SDA*n* is detected low.
		O	When writing as a master or slave, the I²C module drives data on SDA*n* synchronous to SCL*n*.

17.2.2 Detailed Signal Descriptions

SDA*n* and SCL*n*, 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. Refer to the *MPC8349EA Hardware Specifications* for electrical characteristics.

Table 17-2. I²C Interface Signals—Detailed Signal Descriptions

Signal	I/O	Description
SCL1, SCL2	I/O	Serial clock. Performs as an input when the device is programmed as an I²C slave. SCL*n* also performs as an output when the device is programmed as an I²C master.
	O	As outputs for the bidirectional serial clock, these signals operate as described below.
		State Meaning Asserted/Negated—Driven along with SDA*n* as the clock for the data.
	I	As inputs for the bi-directional serial clock, these signals operate as described below.
		State Meaning Asserted/Negated—The I²C unit uses this signal to synchronize incoming data on SDA*n*. The bus is assumed to be busy when this signal is detected low.
SDA1, SDA2	I/O	Serial data. Performs as an input when the device is in a receiving mode. SDA*n* also performs as an output signal when the device is transmitting (as an I²C master or a slave).
	O	As outputs for the bi-directional serial data, these signals operate as described below.
		State Meaning Asserted/Negated—Data is driven.
	I	As inputs for the bi-directional serial data, these signals operate as described below.
		State Meaning Asserted/Negated—Used to receive data from other devices. The bus is assumed to be busy when SDA*n* is detected low.

17.3 Memory Map/Register Definition

Table 17-3 lists the I²C–specific registers and their addresses.

Table 17-3. I²C Memory Map

Address	I²C Register	Access	Reset	Section/Page
0x0_3000	I2C1ADR—I²C1 address register	R/W	0x00	17.3.1.1/17-5
0x0_3004	I2C1FDR—I²C1 frequency divider register	R/W	0x00	17.3.1.2/17-5
0x0_3008	I2C1CR—I²C1 control register	R/W	0x00	17.3.1.3/17-6
0x0_300C	I2C1SR—I²C1 status register	R/W	0x81	17.3.1.4/17-8
0x0_3010	I2C1DR—I²C1 data register	R/W	0x00	17.3.1.5/17-9
0x0_3014	I2C1DFSRR—I²C1 digital filter sampling rate register	R/W	0x10	17.3.1.6/17-10
0x0_301C–0x0_30FF	Reserved, should be cleared	—	—	—
0x0_3100	I2C2ADR—I²C2 address register	R/W	0x00	17.3.1.1/17-5
0x0_3104	I2C2FDR—I²C2 frequency divider register	R/W	0x00	17.3.1.2/17-5
0x0_3108	I2C2CR—I²C2 control register	R/W	0x00	17.3.1.3/17-6
0x0_310C	I2C2SR—I²C2 status register	R/W	0x81	17.3.1.4/17-8
0x0_3110	I2C2DR—I²C2 data register	R/W	0x00	17.3.1.5/17-9
0x0_3114	I2C2DFSRR—I²C2 digital filter sampling rate register	R/W	0x10	17.3.1.6/17-10
0x0_311C–0x0_31FF	Reserved, should be cleared	—	—	—

17.3.1 Register Descriptions

This section describes the I²C 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 I²C*n* data register (I2C*n*DR).

17.3.1.1 I²C*n* Address Register (I2C*n*ADR)

Figure 17-2 shows the I2C*n*ADR register, which contains the address to which the I²C 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 I²C module is in master mode.

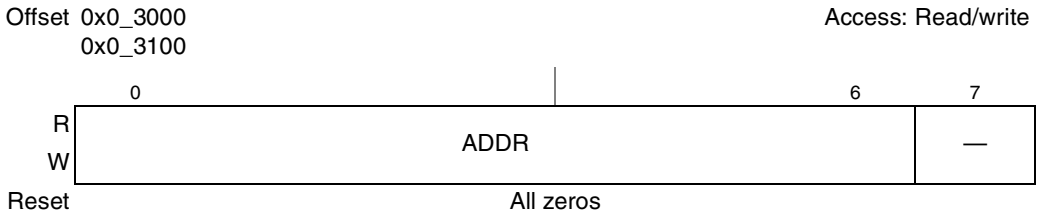


Figure 17-2. I²C*n* Address Register (I2C*n*ADR)

Table 17-4 describes the bit settings of I2C*n*ADR.

Table 17-4. I2C*n*ADR Field Descriptions

Bits	Name	Description
0–6	ADDR	Slave address. Contains the specific slave address that is used by the I²C interface. Note that the default mode of the I²C interface is slave mode for an address match. Note that an address match is one of the conditions that can cause I2C*n*SR[MIF] to be set, signaling an interrupt pending condition.
7	—	Reserved, should be cleared

17.3.1.2 I²C*n* Frequency Divider Register (I2C*n*FDR)

Figure 17-3 shows the bits of the I²C*n* frequency divider register.

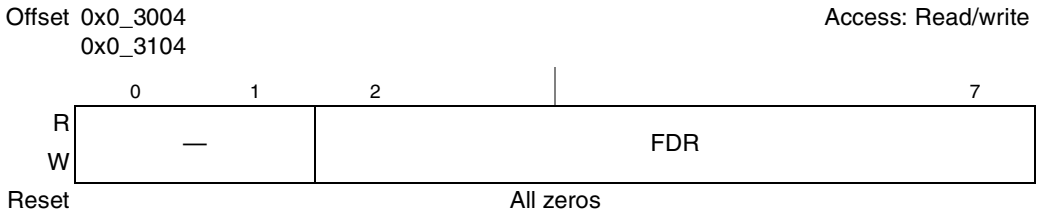


Figure 17-3. I²C*n* Frequency Divider Register (I2C*n*FDR)

Table 17-5 describes the bit settings of I2C*n*FDR. It also maps I2C*n*FDR[FDR] to the clock divider values. Although it describes the ratio between the I²C controller internal clock and SCL, the default ratio of I²C

controller clock and CSB is 1:3 (I²C controller clock frequency is three times slower than CSB clock frequency). This ratio is set in SCCR[TSEC2CM]. Consider this factor when selecting an FDR value.

Table 17-5. I2C*n* FDR Field Descriptions

Bits	Name	Description
0–1	—	Reserved, should be cleared
2–7	FDR	Frequency divider ratio. Used to prescale the clock for bit-rate selection. The serial bit clock frequency of SCL*n* is equal to the I²C*n* controller clock divided by the divider. The serial bit clock frequency divider selections are described as follows:

FDR	Divider (Decimal)	FDR	Divider (Decimal)	FDR	Divider (Decimal)
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 I²C*n* Control Register (I2C*n*CR)

Figure 17-4 shows the I²C*n* control register.

Offset 0x0_3008
0x0_3108

Access: Mixed

	0	1	2	3	4	5	6	7
R	MEN	MIEN	MSTA	MTX	TXAK		—	BCST
W						RSTA		
Reset	All zeros							

Figure 17-4. I²C*n* Control Register (I2C*n*CR)

Table 17-6 describes the I2C*n*CR bit settings.

Table 17-6. I2C*n*CR Field Descriptions

Bits	Name	Description
0	MEN	Module enable. Controls the software reset of the I²C module. 0 The module is reset and disabled. The interface is held in reset, but the registers can still be accessed. 1 The I²C module is enabled. MEN must be set before any other control register bits have any effect. All I²C registers for slave receive or master START can be initialized before setting this bit.
1	MIEN	Module interrupt enable 0 Interrupts from the I²C module are disabled. This does not clear any pending interrupt conditions. 1 Interrupts from the I²C module are enabled. An interrupt occurs provided I2C*n*SR[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 I2C*n*SR[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*n* line during acknowledge cycles for both master and slave receivers. The value of this bit applies only when the I²C 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*n*) 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*n*) 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 I²C to accept broadcast messages at address zero

17.3.1.4 I²C*n* Status Register (I2C*n*SR)

I2CSR is shown in Figure 17-5.

Offset 0x0_300C
0x0_310C

Access: Mixed

	0	1	2	3	4	5	6	7
R	MCF	MAAS	MBB	MAL	BCSTM	SRW	MIF	RXAK
W								
Reset	1	0	0	0	0	0	0	1

Figure 17-5. I²C*n* Status Register (I2C*n*SR)

Table 17-7 describes the bit settings of the I2C*n*SR.

Table 17-7. I2C*n*SR 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: •When I2C*n*DR is read in receive mode or when I2C*n*DR is written in transmit mode •After a start sequence is recognized by the I²C controller in slave mode. 1 Byte transfer is completed
1	MAAS	Addressed as a slave. When the value in I2C*n*ADR matches the calling address or when the calling address is the broadcast address and broadcast mode is enabled (I2C*n*CR[BCST] is set), this bit is set. The processor is interrupted if I2C*n*CR[MIEN] is set. Next, the processor must check the SRW bit and set I2C*n*CR[MTX] accordingly. Writing to the I2C*n*CR 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 I²C bus is idle 1 I²C 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 I2C*n*CR 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 I²C 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: •A complete transfer occurred and no other transfers have been initiated. •The I²C interface is configured as a slave and has an address match. By checking SRW, the processor can select slave transmit/receive mode according to the command of the master.

Table 17-7. I2C*n*SR 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 I2C*n*CR[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: • One byte of data is transferred (set at the falling edge of the 9th clock). • The value in I2C*n*ADR matches with the calling address in slave-receive mode. • Arbitration is lost.
7	RXAK	Received acknowledge. The value of SDA*n* 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 I²C*n* Data Register (I2C*n*DR)

The I2C*n* data register is shown in Figure 17-6.

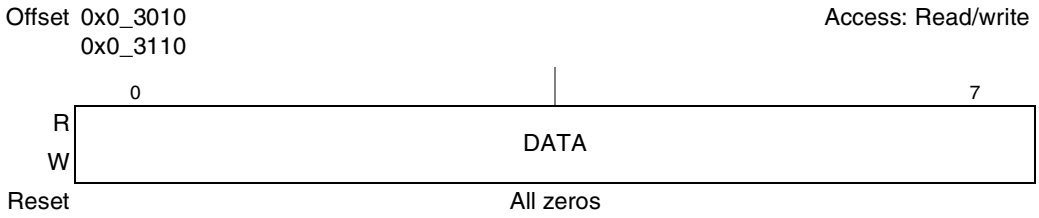


Figure 17-6. I²C*n* Data Register (I2C*n*DR)

Table 17-8 shows the bit descriptions for I2C*n*DR.

Table 17-8. I2C*n*DR Field Description

Bits	Name	Description
0–7	DATA	Transmission starts when an address and the R/W bit are written to the data register and the I²C interface performs as the master. A data transfer is initiated when data is written to the I2C*n*DR. 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 I²C module to receive the next byte of data on the I²C interface. In slave mode, the same function is available after it is addressed. Note that very first read is always a dummy read.

17.3.1.6 Digital Filter Sampling Rate Register (I2C*n*DFSRR)

I2C*n*DFSRR is shown in Figure 17-7.

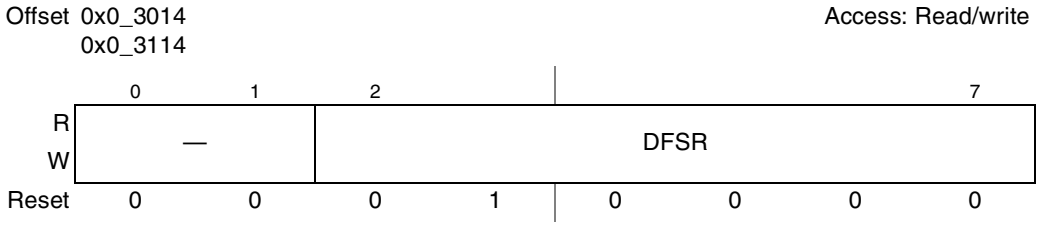


Figure 17-7. I²C*n* Digital Filter Sampling Rate Register (I2C*n*DFSRR)

Table 17-9 shows the I2C*n*DFSRR field descriptions.

Table 17-9. I2C*n*DFSRR 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 I²C bus. The resulting sampling rate is calculated by dividing the platform frequency by the non-zero value of DFSR. If I2C*n*DFSRR is cleared, the I²C bus sample points default to the reset divisor 0x10.

17.4 Functional Description

The I²C 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 I²C interface performs as a slave receiver after the boot sequence has completed.

17.4.1 Transaction Protocol

A standard I²C 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 I²C protocol. The details of the protocol are described in the following subsections.

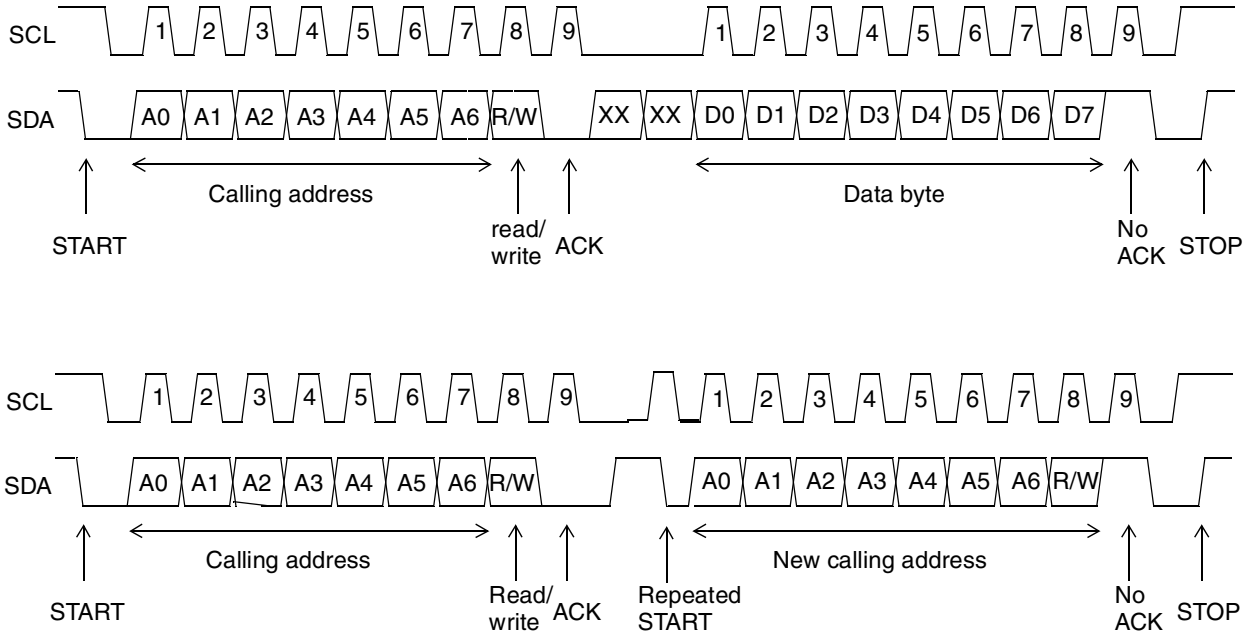


Figure 17-8. I²C Interface Transaction Protocol

17.4.1.1 START Condition

When the I²C bus is not engaged (both SDA*n* and SCL*n* 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*n* while SCL*n* 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 I2C*n*CR[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 I²C module is operating as a master, it must not transmit an address that is the same as its slave address. An I²C 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*n* 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*n* 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 I²C module responds to a general call (broadcast) command when I2C*n*CR[BCST] is set. A broadcast address is always zero; however the I²C module does not check the R/W bit. The second byte of the broadcast message is the master address. Because the second byte is automatically acknowledged by hardware, the receiver device software must verify that the broadcast message is intended for itself by reading the second byte of the message. If the master address is for another receiver device and the third byte is a write command, software can ignore the third byte during the broadcast. If the master address is for another receiver device and the third byte is a read command, software must write 0xFF to I2C*n*DR with I2C*n*CR[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*n* is low and must be held stable while SCL*n* is high, as shown in Figure 17-8. There is one clock pulse on SCL*n* 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*n* 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*n* 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*n* 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*n* signal while SCL*n* 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 I2C*n*CR[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 of how aspects of the protocol are implemented in the I²C module.

17.4.1.5.1 Transaction Monitoring—Implementation Details

The different conditions of the I²C data transfers are monitored as follows (see Figure 17-8):

- START conditions are detected when an SDA*n* fall occurs while SCL*n* is high.
- STOP conditions are detected when an SDA*n* rise occurs while SCL*n* 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 I²C module contains logic that controls the output to the serial data (SDA) and serial clock (SCL) lines of the I²C. The SCL*n* output is pulled low as determined by the internal clock generated in the clock module. The SDA*n* output can change only at the midpoint of a low cycle of the SCL*n*, unless it is performing a START, STOP, or repeated START condition. Otherwise, the SDA*n* output is held constant.

SDA*n* 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*n* signal corresponds to the internal SCL*n* 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 I2C*n*SR
- Whether the module has been addressed as a slave, to update I2C*n*SR and to generate an interrupt
- Whether the address transmitted by the current master matches the general broadcast address

17.4.2 Arbitration Procedure

The I²C 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 I²C 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*n* while another master transmits a logic 0. The losing masters immediately switch to slave-receive mode and stop driving the SDA*n* line. In this case, the transition from master to slave mode does not generate a STOP condition. Meanwhile, the I²C unit sets I2C*n*SR[MAL] to indicate the loss of arbitration and, as a slave, services the transaction if it is directed to itself.

If the I²C module is enabled in the middle of an ongoing byte transfer, the interface behaves as follows:

- Slave mode—the I²C module ignores the current transfer on the bus and starts operating whenever a subsequent START condition is detected.
- Master mode—the I²C 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*n* 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 I2C*n*SR[MAL] is set) under the following conditions:

- SDA*n* samples low when the master drives high during an address or data-transmit cycle (transmit).
- SDA*n* 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 I²C 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*n* 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*n* 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*n* line, a high-to-low transition on the SCL*n* line affects all devices connected on the bus. The devices begin counting their low period when the master negates the SCL*n* line. After a device has negated SCL*n*, it holds the SCL*n* 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*n* if another device is still within its low period. Therefore, SCL*n* 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*n* is released and asserted. Then there is no difference between the devices' clocks and the state of SCL*n*, and all the devices begin counting their high periods. The first device to complete its high period negates SCL*n* again.

17.4.4.2 Input Synchronization and Digital Filter

The following sections describes synchronization of the input signals and the filtering of SCL*n* and SDA*n* in detail.

17.4.4.2.1 Input Signal Synchronization

The input synchronization block synchronizes the input SCL*n* and SDA*n* signals to the system clock and detects transitions of these signals.

17.4.4.2.2 Filtering of SCL*n* and SDA*n* Lines

The SCL*n* and SDA*n* inputs are filtered to eliminate noise. Three consecutive samples of the SCL*n* and SDA*n* 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*n* low, the slave can drive SCL*n* low for the required period and then release it. If the slave SCL*n* low period is greater than the master SCL*n* low period, the resulting SCL*n* 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 I²C module communicates with one or more EEPROMs through the I²C interface. EEPROMs can be programmed to initialize one or more configuration registers. Note that as described in Section 4.3.2.2.3, “Boot Sequencer Configuration,” the default value for BOOTSEQ is 0b00, which corresponds to the I²C boot sequencer being disabled at power-up.

Boot sequencer mode also supports an extension of the standard I²C 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 4.3.2.2.3, “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 I²C interface is used, the I²C 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 I²C controller to hang. The I²C 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 I²C 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 I²C boot sequencer. See Section 4.3.2.2.3, "Boot Sequencer Configuration."

Note that this usage does not prevent using the I²C boot sequencer to initiate the device in the normal functional mode, after reset state has completed. However, an I²C 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 I²C 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 0xAA55AA. The I²C 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 the 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 IMMRRBAR 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 ALTCBAR 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 to use one of the GPIO signals 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 I²C interface. Figure 17-11 is a recommended flowchart for I²C interrupt service routines.

A **sync** assembly instruction must be executed after each I²C register read/write access to guarantee that register accesses occur in order.

The I²C controller does not guarantee its recovery from all illegal I²C 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 I²C bus hangs. The recovery routine should also handle the case when the illegal I²C 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 I²C interrupt service routine. Deviation from the flowchart may result in unpredictable I²C bus behavior. However, in the slave receive mode (not shown), the interrupt service routine may need to set I2C*n*CR[TXAK] when the next-to-last byte is to be accepted. It is recommended that a **sync** instruction follow each I²C register read or write to guarantee that register accesses occur in order.

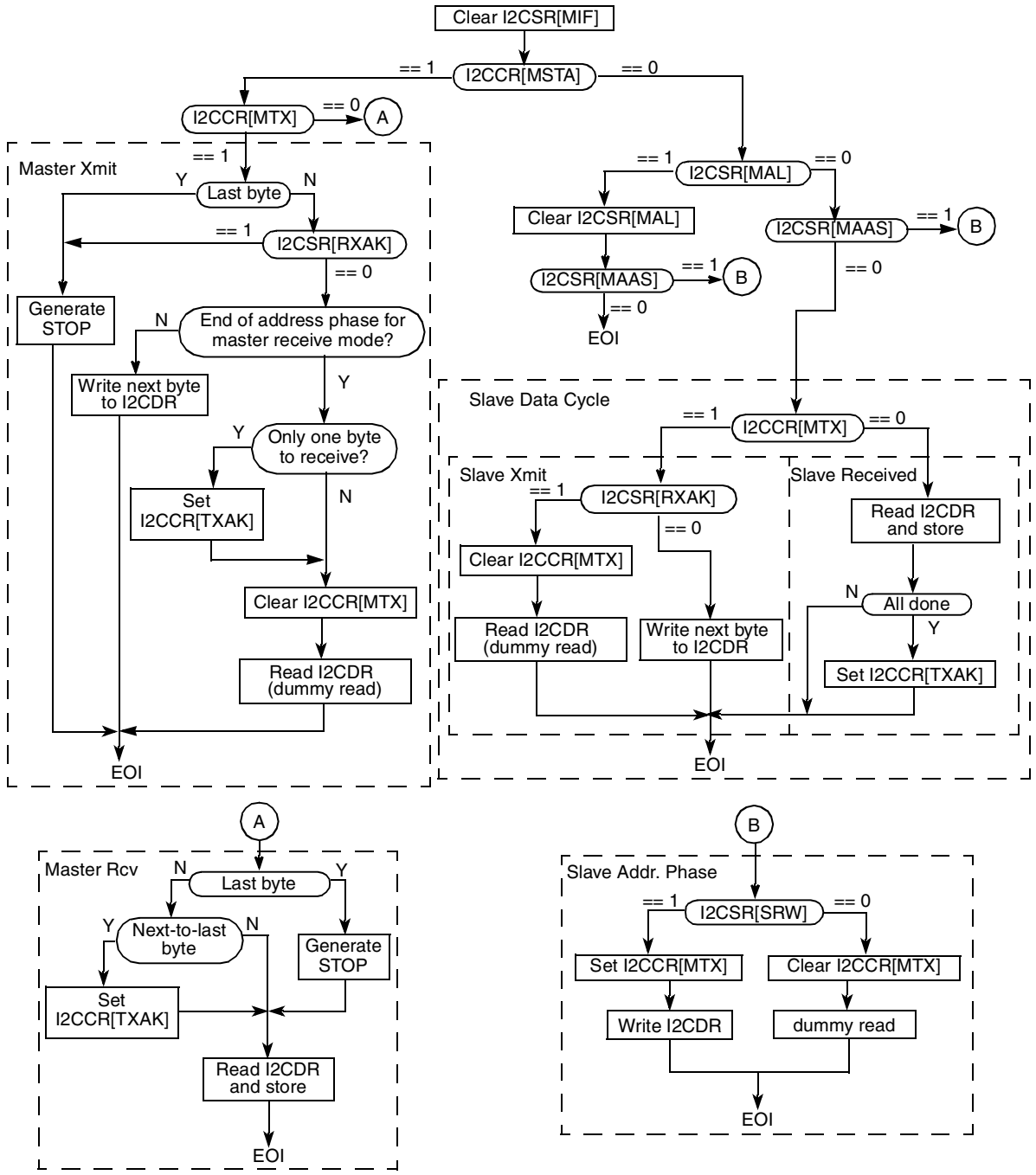


Figure 17-11. Example I²C Interrupt Service Routine Flowchart

17.5.2 Initialization Sequence

A hard reset initializes all of the I²C registers to their default states. The following initialization sequence initializes the I²C unit:

1. All I²C registers must be located in a cache-inhibited page.
2. Update I2C*n*FDR[FDR] and select the required division ratio to obtain the SCL*n* frequency from the CSB (platform) clock.
3. Update I2C*n*ADR to define the slave address for this device.
4. Modify I2C*n*CR to select master/slave mode, transmit/receive mode, and interrupt-enable or disable.
5. Set the I2C*n*CR[MEN] to enable the I²C 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 I²C system, check whether the serial bus is free (I2C*n*SR[MBB] = 0) before switching to master mode.
2. Select master mode (set I2C*n*CR[MSTA]) to transmit serial data and select transmit mode (set I2C*n*CR[MTX]) for the address cycle.
3. Write the slave address being called into I2C*n*DR. The data written to I2C*n*DR[0–6] comprises the slave calling address. I2C*n*CR[MTX] indicates the direction of transfer (transmit/receive) required from the slave.

The scenario above assumes that the I²C interrupt bit (I2C*n*SR[MIF]) is cleared. If MIF is set at any time, an I²C interrupt is generated (provided interrupt reporting is enabled with I2C*n*CR[MIEN] =1).

17.5.4 Post-Transfer Software Response

Transmission or reception of a byte automatically sets the data transferring bit (I2C*n*SR[MCF]), which indicates that one byte has been transferred. The I²C interrupt bit (I2C*n*SR[MIF]) is also set and an interrupt is generated to the processor if the interrupt function is enabled during the initialization sequence (I2C*n*CR[MIEN] is set). In the interrupt handler, software must take the following steps:

1. Clear I2C*n*SR[MIF]
2. Read the I2C*n*DR in receive mode or write to I2C*n*DR in transmit mode. Note that this causes I2C*n*SR[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, I2C*n*CR[MTX] must be toggled at this stage (see Figure 17-11).

If the interrupt function is disabled, software can service the I2C*n*DR in the main program by monitoring I2C*n*SR[MIF]. In this case, I2C*n*SR[MIF] must be polled rather than I2C*n*SR[MCF] because MCF behaves differently when arbitration is lost. Note that interrupt or other bus conditions may be detected before the I²C signals have time to settle. Thus, when polling I2C*n*SR[MIF] (or any other I2C*n*SR bits), software delays may be needed to give the I²C signals sufficient time to settle.

During slave-mode address cycles (I2C*n*SR[MAAS] is set), I2C*n*SR[SRW] should be read to determine the direction of the subsequent transfer and I2C*n*CR[MTX] should be programmed accordingly. For slave-mode data cycles (MAAS is cleared), I2C*n*SR[SRW] is not valid and I2C*n*CR[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 (I2C*n*CR[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 I²C interface, so the last byte does not receive the data acknowledge (because I2C*n*CR[TXAK] is set). For 1-byte transfers, a dummy read should be performed by the interrupt service routine (see Figure 17-11). 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 I2C*n*CR[RSTA].

17.5.7 Generation of SCL*n* when SDA*n* is Negated

It is sometimes necessary to force the I²C module to become the I²C bus master out of reset and drive SCL*n* (even though SDA*n* may already be driven, which indicates that the bus is busy). This can occur when a system reset does not cause all I²C devices to be reset. Thus, SDA*n* can be negated low by another I²C device while this I²C module is coming out of reset and will stay low indefinitely. The following procedure can be used to force this I²C module to generate SCL*n* so that the device driving SDA*n* can finish its transaction:

1. Disable the I²C module and set the master bit by setting I2C*n*CR to 0x20.
2. Enable the I²C module by setting I2C*n*CR to 0xA0.
3. Read I2C*n*DR.
4. Return the I²C module to slave mode by setting I2C*n*CR 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 I2C*n*SR[MAAS] is set, software should set the transmit/receive mode select bit (I2C*n*CR[MTX]) according to the R/$\overline{W}$ command bit (I2C*n*SR[SRW]). Writing to I2C*n*CR 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 I2C*n*DR for slave transmits or dummy reading from

I2C*n*DR in slave-receive mode. The slave negates SCL*n* between byte transfers. SCL*n* is released when I2C*n*DR is accessed in the required mode.

17.5.8.1 Slave Transmitter and Received Acknowledge

In the slave transmitter routine, the received acknowledge bit (I2C*n*SR[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 (I2C*n*SR[RXAK] is set), the slave transmitter interrupt routine must clear I2C*n*CR[MTX] to switch the slave from transmitter to receiver mode. A dummy read of I2C*n*DR then releases SCL*n* 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:

- I2C*n*SR[MAL] is set
- I2C*n*CR[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 I2C*n*SR[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 MPC8349EA. 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
- Clear to send ($\overline{\text{CTS}}$) input port and request to send ($\overline{\text{RTS}}$) output port for data-flow control.
- 16-bit counter for baud rate generation
- Interrupt control logic

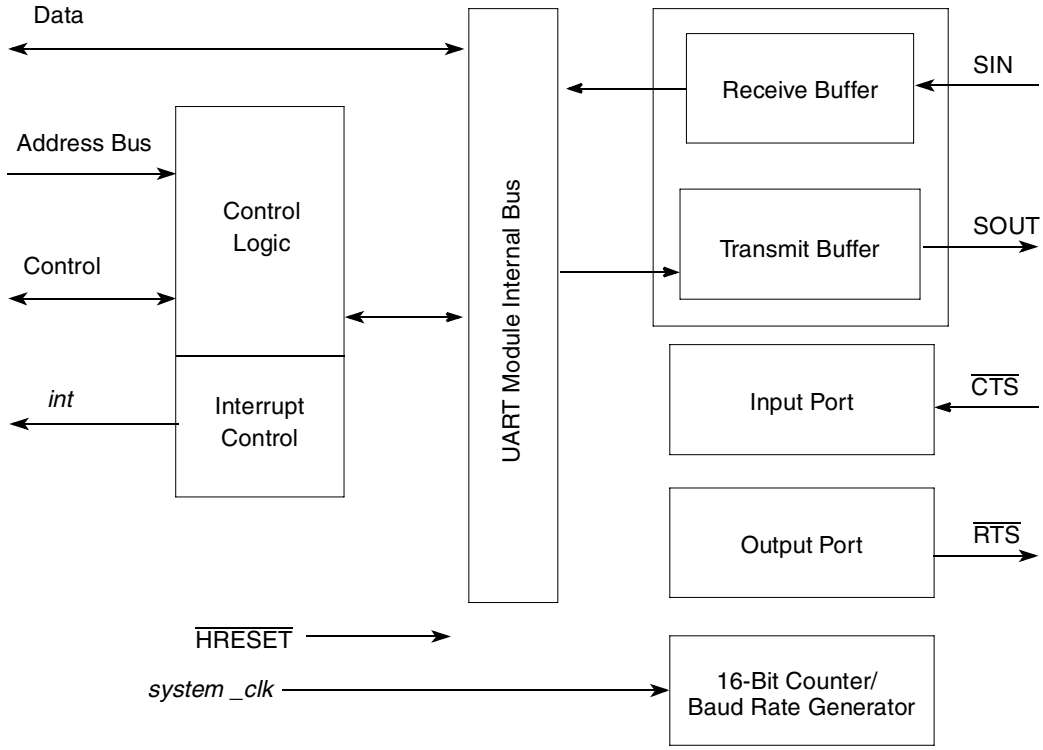


Figure 18-1. UART Block Diagram

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
- Clear-to-send ($\overline{\text{CTS}}$) and ready-to-send ($\overline{\text{RTS}}$) MODEM control functions.
- Software-selectable serial interface data format (data length, parity, 1/1.5/2 STOP bit, baud rate)
- Line and MODEM status registers
- Line-break detection and generation
- Internal diagnostic support, local loopback, and break functions
- Prioritized interrupt reporting
- Overrun, parity, and framing error detection

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), 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
$\overline{\text{UART_CTS}}$[1:2]	I	2	1	Clear to send UART1 and UART2
$\overline{\text{UART_RTS}}$[1:2]	O	2	1	Request to send 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]	I	Serial data in. Data is received on the receivers of UART1 and UART2 through its respective serial data input signal, with the least significant bit received first.	
		State Meaning	Asserted/Negated—Represents the data being received on the UART interface.
		Timing	Assertion/Negation—An internal logic sample signal, *rxcnt*, uses the frequency of the baud-rate generator to sample the data on SIN.

Table 18-2. DUART Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
UART_SOUT[1:2]	O	Serial data out. The serial data output signals for the UART1 and UART2 are set (mark condition) when the transmitter is disabled, idle, or operating in the local loop-back mode. Data is shifted out on these signals, with the least significant bit transmitted first.	
		State Meaning	Asserted/Negated—Represents the data transmitted on the respective UART interface.
		Timing	Assertion/Negation—An internal logic sample signal, *rxcnt*, uses the frequency of the baud-rate generator to update and drive the data on SOUT.
$\overline{\text{UART_CTS}}$[1:2]	I	Clear to send. Connected to the respective $\overline{\text{RTS}}$ outputs of the other UART devices on the bus. They can be programmed to generate an interrupt on change-of-state of the signal.	
		State Meaning	Asserted/Negated—Represent the clear to send condition for their respective UART.
		Timing	Assertion/Negation—Sampled at the rising edge of every system clock.
$\overline{\text{UART_RTS}}$[1:2]	O	Request to send. $\overline{\text{C}}$an be programmed to be automatically negated and asserted by either the receiver or transmitter. When connected to the $\overline{\text{CTS}}$ input of a transmitter, this signal can be used to control serial data flow.	
		State Meaning	Asserted/Negated—Represents the data being transmitted on the respective UART interface.
		Timing	Assertion/Negation—Updated and driven at the rising edge of every system clock.

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
0x0_4500	URBR—ULCR[DLAB] = 0 UART1 receiver buffer register	R	0x00	18.3.1.1/18-6
	UTHR—ULCR[DLAB] = 0 UART1 transmitter holding register	W	0x00	18.3.1.2/18-6
	UDLB—ULCR[DLAB] = 1 UART1 divisor least significant byte register	R/W	0x00	18.3.1.3/18-7
0x0_4501	UIER—ULCR[DLAB] = 0 UART1 interrupt enable register	R/W	0x00	18.3.1.4/18-8
	UDMB—ULCR[DLAB] = 1 UART1 divisor most significant byte register	R/W	0x00	18.3.1.3/18-7
0x0_4502	UIIR—ULCR[DLAB] = 0 UART1 interrupt ID register	R	0x01	18.3.1.5/18-9
	UFCR—ULCR[DLAB] = 0 UART1 FIFO control register	W	0x00	18.3.1.6/18-10
	UAFR—ULCR[DLAB] = 1 UART1 alternate function register	R/W	0x00	18.3.1.12/18-16
0x0_4503	ULCR—ULCR[DLAB] = x UART1 line control register	R/W	0x00	18.3.1.7/18-11
0x0_4504	UMCR—ULCR[DLAB] = x UART1 MODEM control register	R/W	0x00	18.3.1.8/18-13
0x0_4505	ULSR—ULCR[DLAB] = x UART1 line status register	R	0x60	18.3.1.9/18-14
0x0_4506	UMSR—ULCR[DLAB] = x UART1 MODEM status register	R	0x00	18.3.1.10/18-15
0x0_4507	USCR—ULCR[DLAB] = x UART1 scratch register	R/W	0x00	18.3.1.11/18-16
0x0_4510	UDSR—ULCR[DLAB] = x UART1 DMA status register	R	0x01	18.3.1.13/18-17
0x0_4600	URBR—ULCR[DLAB] = 0 UART2 receiver buffer register	R	0x00	18.3.1.1/18-6
	UTHR—ULCR[DLAB] = 0 UART2 transmitter holding register	W	0x00	18.3.1.2/18-6
	UDLB—ULCR[DLAB] = 1 UART2 divisor least significant byte register	R/W	0x00	18.3.1.3/18-7
0x0_4601	UIER—ULCR[DLAB] = 0 UART2 interrupt enable register	R/W	0x00	18.3.1.4/18-8
	UDMB—ULCR[DLAB] = 1 UART2 divisor most significant byte register	R/W	0x00	18.3.1.3/18-7
0x0_4602	UIIR—ULCR[DLAB] = 0 UART2 interrupt ID register	R	0x01	18.3.1.5/18-9
	UFCR—ULCR[DLAB] = 0 UART2 FIFO control register	W	0x00	18.3.1.6/18-10
	UAFR—ULCR[DLAB] = 1 UART2 alternate function register	R/W	0x00	18.3.1.12/18-16
0x0_4603	ULCR—ULCR[DLAB] = x UART2 line control register	R/W	0x00	18.3.1.7/18-11
0x0_4604	UMCR—ULCR[DLAB] = x UART2 MODEM control register	R/W	0x00	18.3.1.8/18-13
0x0_4605	ULSR—ULCR[DLAB] = x UART2 line status register	R	0x60	18.3.1.9/18-14
0x0_4606	UMSR—ULCR[DLAB] = x UART2 MODEM status register	R	0x00	18.3.1.10/18-15
0x0_4607	USCR—ULCR[DLAB] = x UART2 scratch register	R/W	0x00	18.3.1.11/18-16
0x0_4610	UDSR—ULCR[DLAB] = x UART2 DMA status register	R	0x01	18.3.1.13/18-17

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 UTHRs.

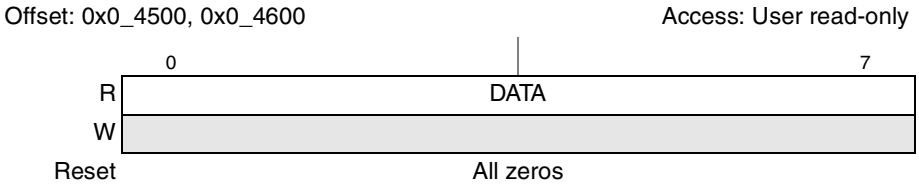


Figure 18-2. Receiver Buffer Registers (URBR1 and URBR2)

Table 18-4 describes URBR.

Table 18-4. URBR Field Description

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 18-21 and Table 18-22.

Figure 18-3 shows the bits in the UTHRs.

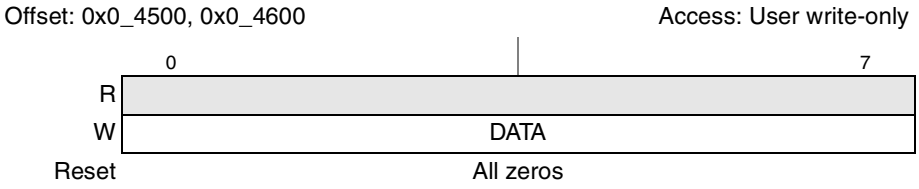


Figure 18-3. Transmitter Holding Registers (UTHR1 and UTHR2)

Table 18-5 describes the UTHR.

Table 18-5. UTHR Field Description

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 x [UDMB||UDLB]). Equivalently, þþ [UDMB||UDLB:0b0000]þ = 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.

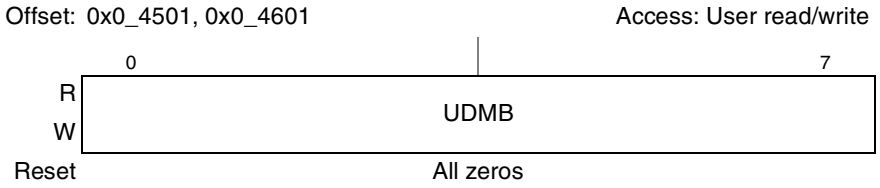


Figure 18-4. Divisor Most Significant Byte Registers (UDMB1 and UDMB2)

Table 18-6 describes the UDMB.

Table 18-6. UDMB Field Description

Bits	Name	Description
0–7	UDMB	Divisor most significant byte

Figure 18-5 shows the bits in the UDLBs.

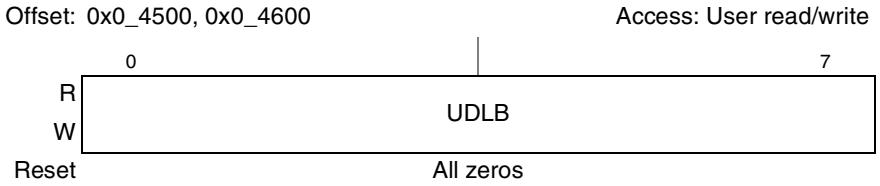


Figure 18-5. Divisor Least Significant Byte Registers (UDLB1 and UDLB2)

Table 18-7 describes the UDLB.

Table 18-7. UDLB Field Description

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	1736	6C8	266	0.0064
19,200	868	364	266	0.0064
38,400	434	1B2	266	0.0064
56,000	298	12A	266	0.1280
128,000	130	82	266	0.1600
256,000	65	41	266	0.1600
9,600	2170	87A	333	0.0064
19,200	1085	43D	333	0.0064
38,400	543	21F	333	0.0858
56,000	372	174	333	0.0064
128,000	163	A3	333	0.1472
256,000	81	51	333	0.4672

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 x 16).
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:

   Percent error value = (1 - AFI/ICF) x 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.

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 (UIIR1 and UIIR2)

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.

Offset: 0x0_4502, 0x0_4602 Access: User read-only

	0	1	2	3	4	5	6	7
R	FE		—		IID3	IID2	IID1	IID0
W								
Reset	0	0	0	0	0	0	0	1

Figure 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
0000	Fourth	MODEM status	$\overline{\text{CTS}}$ input value changed since last read of UMSR.	Reading UMSR

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.

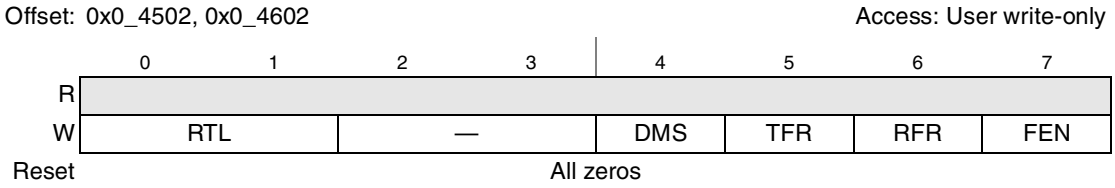


Figure 18-8. FIFO Control Registers (UFCR1 and UFCR2)

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 Section 18.4.5.2, “DMA Mode Select”. 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.

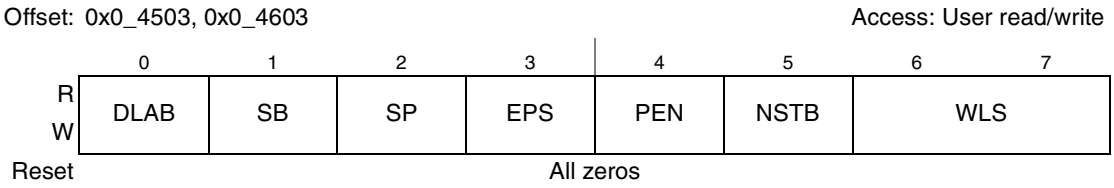


Figure 18-9. Line Control Register (ULCR1 and ULCR2)

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 Table 18-14. 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	NTSB	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

	0	1	2	3	4	5	6	7
R W	—			LOOP	—		RTS	—
Reset	All zeros							

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 loop-back mode 0 Normal operation. 1 Functionally, the data written to UTHR can be read from URBR of the same UART, and UMCR[RTS] is tied to UMSR[CTS].
4–5	—	Reserved
6	RTS	Ready to send 0 Negates corresponding $\overline{\text{UART_RTS}}$ output. 1 Assert corresponding $\overline{\text{UART_RTS}}$ output. Informs external MODEM or peripheral that the UART is ready for sending/receiving data.
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

	0	1	2	3	4	5	6	7
R	RFE	TEMT	THRE	BI	FE	PE	OE	DR
W								
Reset	0	1	1	0	0	0	0	0

Figure 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 MODEM Status Registers (UMSR1 and UMSR2)

The UMSRs, shown in Figure 18-12, track the status of the MODEM (or external peripheral device) $\overline{\text{CTS}}$, set for the corresponding UART.

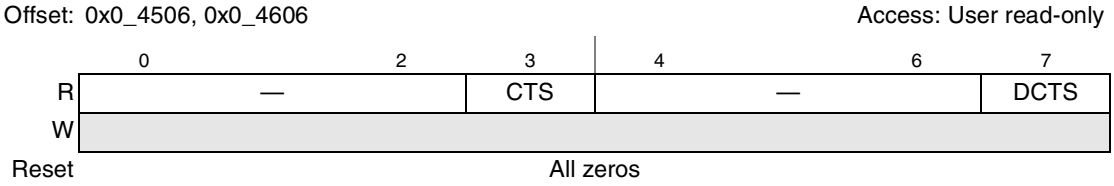


Figure 18-12. Modem Status Register (UMSR1 and UMSR2)

Table 18-17 describes UMSR fields.

Table 18-17. UMSR Field Descriptions

Bits	Name	Description
0–2	—	Reserved, should be cleared
3	CTS	Clear to send. Represents the inverted value of the $\overline{\text{CTS}}$ input pin from the external peripheral device. 0 Corresponding $\overline{\text{CTS}}n$ is negated. 1 Corresponding $\overline{\text{CTS}}n$ is asserted. The MODEM or peripheral device is ready for data transfers.
4–6	—	Reserved, should be cleared
7	DCTS	Delta clear to send 0 No change on the corresponding $\overline{\text{CTS}}n$ signal since the last read of UMSR[CTS]. 1 $\overline{\text{CTS}}n$ changed since the last read of UMSR[CTS]. Causes an interrupt if UIER[EMSI] is set to detect this condition.

18.3.1.11 Scratch Registers (USCR1 and USCR2)

USCR, shown in Figure 18-13, are for debugging software or the DUART hardware. The USCRs do not affect the operation of the DUART.

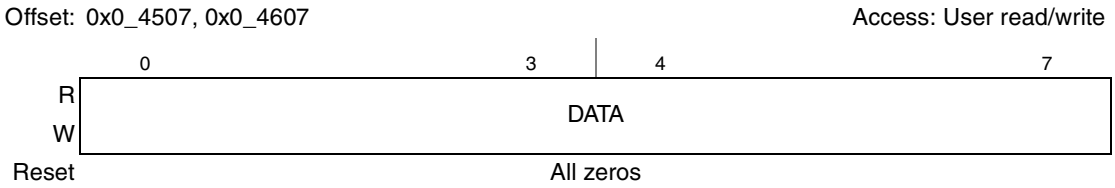


Figure 18-13. Scratch Register (USCR)

Table 18-18 describes USCR fields.

Table 18-18. USCR Field Descriptions

Bits	Name	Description
0–7	DATA	Data

18.3.1.12 Alternate Function Registers (UAFR1 and UAFR2)

The UAFRs, shown in Figure 18-14, 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.

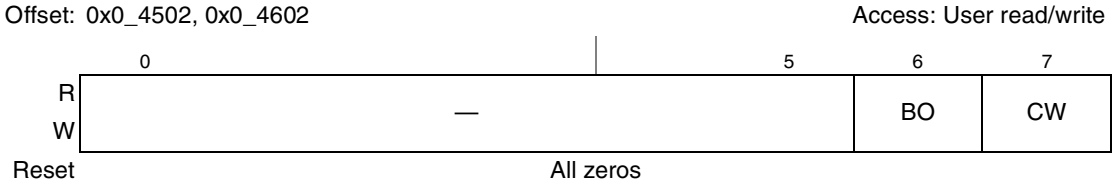


Figure 18-14. Alternate Function Register (UAFR)

Table 18-19 describes UAFR fields.

Table 18-19. 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.

18.3.1.13 DMA Status Registers (UDSR1 and UDSR2)

The DMA status registers (UDSRs), shown in Figure 18-15, return transmitter and receiver FIFO status and provide the ability to assist DMA data operations to and from the FIFOs.

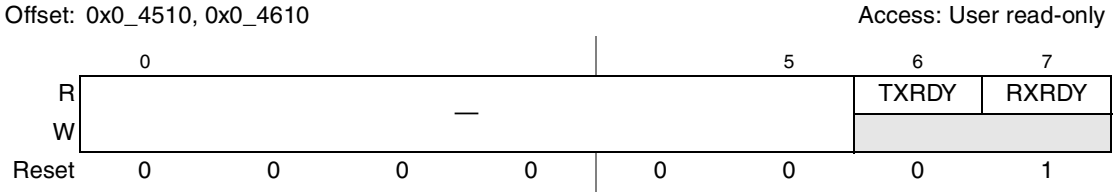


Figure 18-15. DMA Status Register (UDSR)

Table 18-20 describes the fields of the UDSRs.

Table 18-20. 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 inTable 18-22. 1 This bit is set, as shown in Table 18-21.
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 18-24. 1 This bit is set, as shown in Table 18-23.

Table 18-21. 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 18-22. 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 18-23. 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 18-24. 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

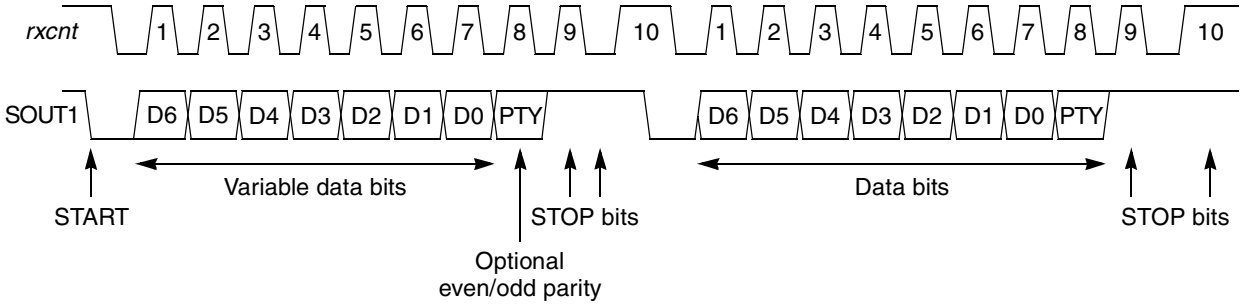
The communication channel provides a full-duplex asynchronous receiver and transmitter using an operating frequency derived from the system clock signal.

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 18-16. Therefore, only two devices are attached to the same signals and there is no need for address or arbitration bus cycles.



Two 7-bit data transmissions with parity and 2-bit STOP transactions

Figure 18-16. 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 loop-back 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 18-16 shows that the START bit is defined as a logic 0. The START bit denotes the beginning of a new data transfer which is limited to the bit length programmed in 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:

Baud rate = (1/16) x (system clock frequency/divisor value)

Therefore, the output frequency of the baud-rate generator is 16 times the baud rate.

The divisor value is determined by the following two 8-bit registers to form a 16-bit binary number:

- UART divisor most significant byte register (UDMB)
- UART divisor least significant byte register (UDLB)

Upon loading either of the divisor latches, a 16-bit baud-rate counter is loaded.

The divisor latches must be loaded during initialization to ensure proper operation of the baud-rate generator. Both UART devices on the same bus must be programmed for the same baud rate before starting a transfer.

The baud clock can be passed to the performance monitor by enabling UAFR[BO]. This can be used to determine baud-rate errors.

18.4.3 Local Loop-Back Mode

Local loop-back mode is provided for diagnostic testing. The data written to UTHR can be read from the receiver buffer register (URBR) of the same UART. In this mode, the MODEM control register UMCR[RTS] is internally tied to the MODEM status register UMSR[CTS]. The transmitter SOUT is set to a logic 1 and the receiver SIN is disconnected. The output of the transmitter shift register is looped back into the receiver shift register input. The $\overline{\text{CTS}}$ (input signal) is disconnected, $\overline{\text{RTS}}$ is internally connected to $\overline{\text{CTS}}$, and the $\overline{\text{RTS}}$ (output signal) becomes inactive. In this diagnostic mode, data that is transmitted is immediately received. In local loop-back mode the transmit and receive data paths of the DUART can be verified. Note that in local loop-back 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]) 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 18.3.1.13, “DMA Status Registers (UDSR1 and UDSR2),” for a complete description of the USDR[RXRDY] and USDR[TXRDY] bits.

18.4.5.3 Interrupt Control Logic

An interrupt is active when DUART interrupt ID register bit 0 (UIIR[0]), 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[0] to determine whether the UART is ready for service. Software must monitor the appropriate ULSR and UMSR bits. UIIR[0] 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 are 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, the 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 Figure 19-1, giving 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.

19.2 Introduction

The SPI block diagram is shown in Figure 19-1.

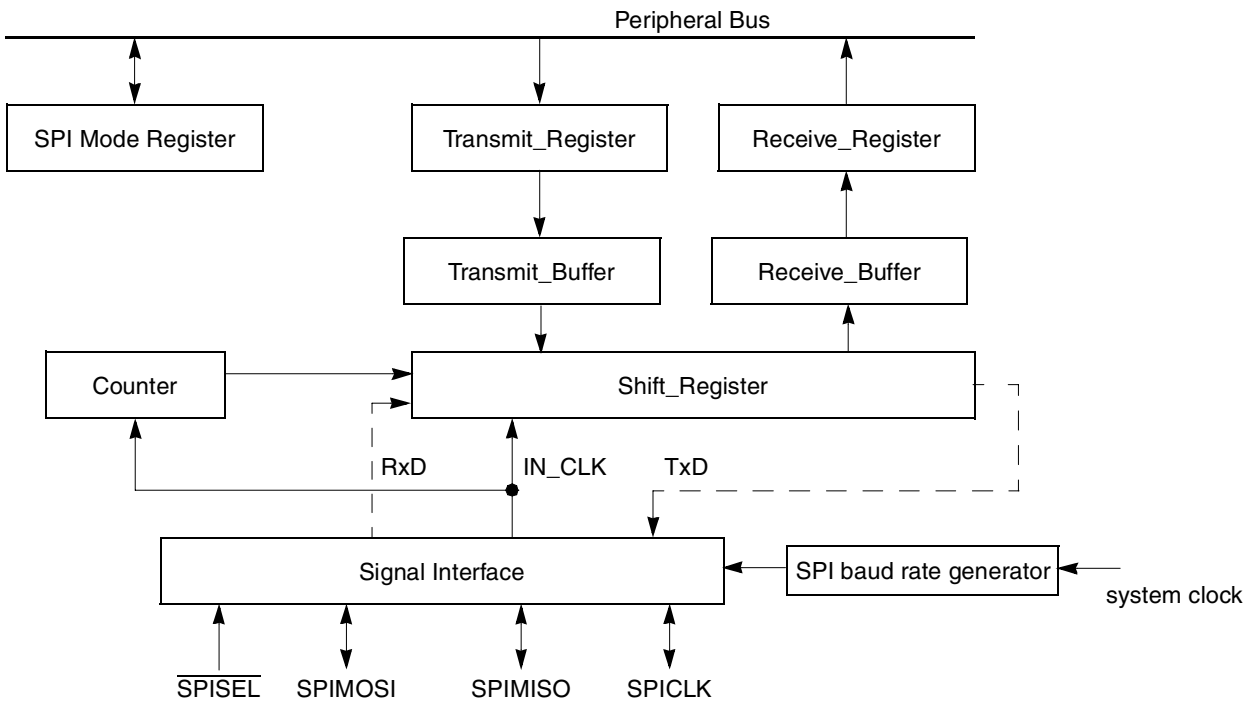


Figure 19-1. SPI Block Diagram

19.2.1 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 (system clock rate/4) in master mode; (system 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

19.2.2 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 19.4.1.4, “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.

19.2.3 Modes of Operation

The SPI can be programmed to work in a single- or multiple-master environment. This section describes SPI master and slave operations in a single-master configuration. It also discusses the multiple master environment.

The following sections summarize the main modes of operation that the SPI supports.

19.2.3.1 The SPI as a Master Device

In master mode, the SPI sends a message to the slave peripheral, which sends back a simultaneous reply. A single master device with multiple slaves can use general-purpose parallel I/O signals to selectively enable slaves, as shown in Figure 19-2. To eliminate the multi-master error in a single-master environment, the master’s $\overline{\text{SPISEL}}$ input should be forced inactive by an external pull up.

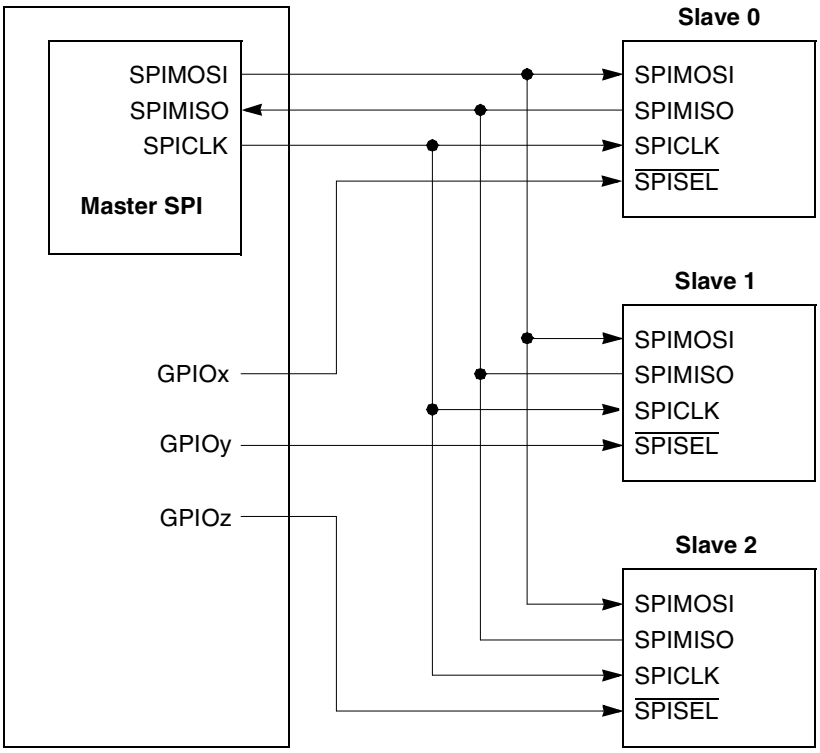


Figure 19-2. Single-Master/Multi-Slave Configuration

To start exchanging data, the processor core writes the data to be sent into the SPITD register. The SPI then generates programmable clock pulses on SPICLK for each character. It shifts Tx data out on the SPI master-out slave-in (SPIMOSI) and Rx data in on the SPI master-in slave-out (SPIMISO) simultaneously. During transmission, the core is responsible for supplying the data whenever the SPI requests it to ensure smooth operation. After the last data (LST command and data afterwards), the first character written to SPITD acts as a start command for the SPI.

The SPI continues transmitting and receiving characters until SPCOM[LST] is set or an error occurs.

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.

19.2.3.2 The SPI as a Slave Device

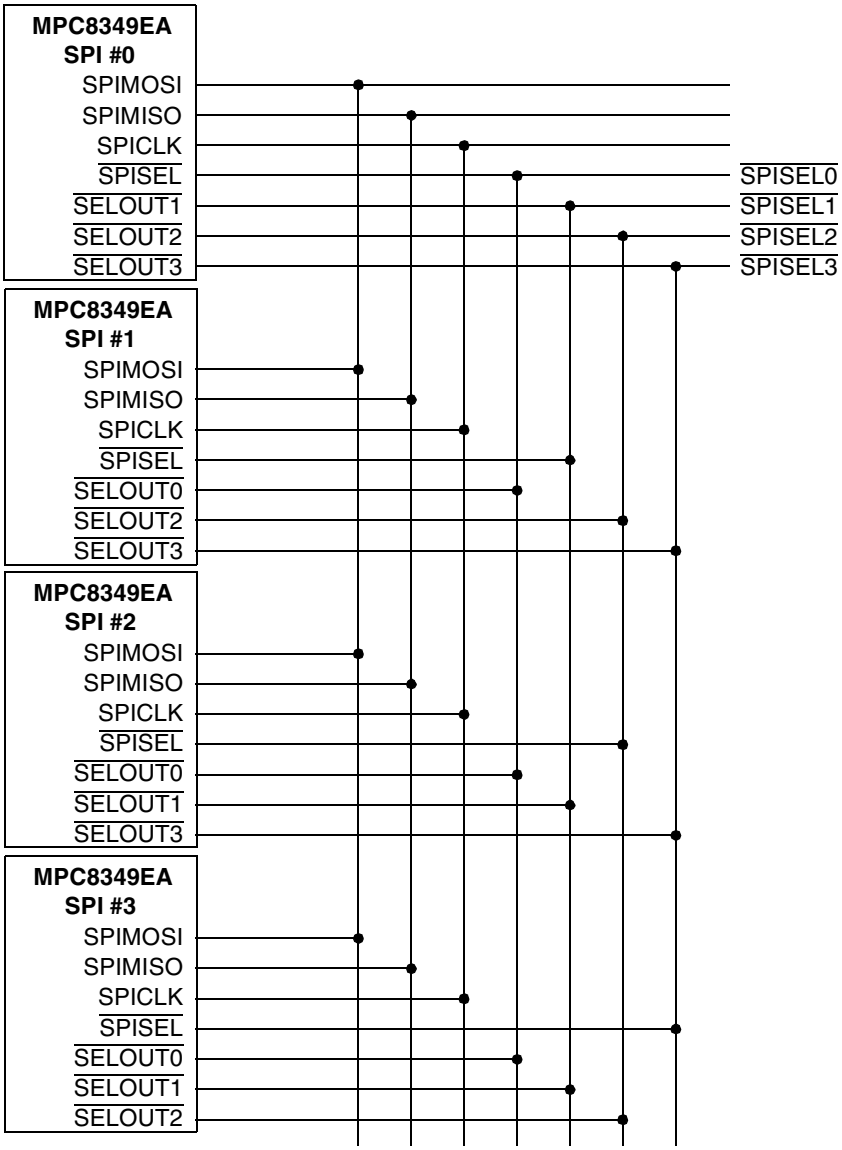
In slave mode, the SPI receives messages from an SPI master and sends a simultaneous reply. The slave's $\overline{\text{SPISEL}}$ must be asserted before Rx clocks are recognized. Once $\overline{\text{SPISEL}}$ is asserted, SPICLK becomes an input from the master to the slave. SPICLK can be any frequency from DC to system 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.

19.2.3.3 The 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, 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. 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 system (csb) clock/50. However, the SPI can transfer a single character at much higher rates—system clock/4 in master mode and system clock/2 in slave mode. Gaps should be inserted between multiple characters to keep from exceeding the maximum sustained data rate.

19.3 External Signal Descriptions

The SPI's four wire interface consists of transmit, receive, clock and slave select.

19.3.1 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
$\overline{\text{SPISEL}}$	SPI slave select	—	Required in open drain mode

19.3.2 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	
		State Meaning	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
		Timing	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	
		State Meaning	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
		Timing	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	
		State Meaning	Assertion/Negation according to SPMODE[PM,DIV16] register rate configuration
		Timing	Assertion/Negation—during frame reception/transmission
$\overline{\text{SPISEL}}$	I	SPI slave select	
		State Meaning	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.
		Timing	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 system 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 by 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.

19.4 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 of CCSRBAR together with the SPI block base address and offset listed in Table 19-3. Undefined 4-byte address spaces within offset 0x000–0xFFF are reserved.

Table 19-3. SPI Register Summary

Offset	Register	Access	Reset Value	Section/Page
0x000–0x01F	Reserved	—	—	—
0x020	SPI mode register (SPMODE)	R/W	0x0000_0000	19.4.1.1/19-9
0x024	SPI event register (SPIE)	Mixed	0x0000_0000	19.4.1.2/19-12
0x028	SPI mask register (SPIM)	R/W	0x0000_0000	19.4.1.2/19-12
0x02C	SPI command register (SPCOM)	R/W	0x0000_0000	19.4.1.4/19-14
0x030	SPI transmit register (SPITD)	R/W	0x0000_0000	19.4.1.5/19-14
0x034	SPI receive register (SPIRD)	R	0xFFFF_FFFF	19.4.1.6/19-15
0x038–0xFFF	Reserved	—	—	—

19.4.1 Register Descriptions

19.4.1.1 SPI Mode Register (SPMODE)

SPMODE, shown in Figure 19-4, controls both the SPI operation mode and clock source.

Offset 0x020 Access: Read/write

	0	1	2	3	4	5	6	7	8 11	12 15	16 18	19	20 31
R W	—	LOOP	CI	CP	DIV16	REV	M/S	EN	LEN	PM	—	OD	—
Reset	All zeros												

Figure 19-4. SPMODE-SPI Mode Register

Table 19-4 describes the SPMODE fields.

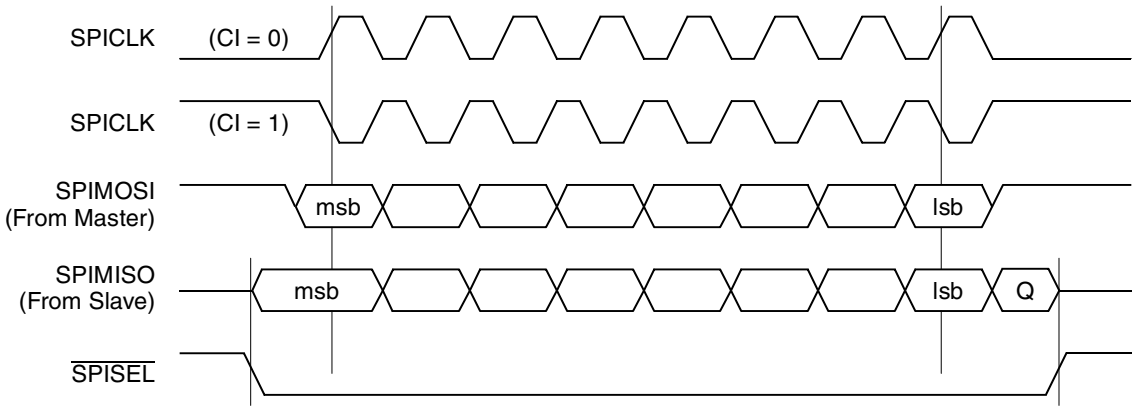
Table 19-4. SPMODE Field Descriptions

Bits	Name	Description
0	—	Reserved. Should be cleared.
1	LOOP	Loop mode. Enables local loopback operation. 0 Normal operation. 1 Loopback mode. Used to test the SPI controller internal functionality, the transmitter output is internally connected to the receiver input. The receiver and transmitter operate normally, except that received data is ignored. The SPI acts normally in loop back mode; therefore, negating SPISEL in slave mode stops transmission, negating it in master mode and causing and MME error.
2	CI	Clock invert. Inverts SPI clock polarity. See Figure 19-5 and Figure 19-6 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 Figure 19-5 and Figure 19-6 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.

Table 19-4. SPMODE Field Descriptions (continued)

Bits	Name	Description
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 System clock is the input to the SPI BRG. 1 System 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 Section 19.4.1.6.1, “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. **Note:** The SPI controller requires a minimal gap of at least 10 system 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-bits 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 that the valid bits will be16-20. 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 system clock or system clock divided by 16, depending on DIV16 bit) is divided by 4 * ([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*(4*(0011+1))=256. In slave mode this field must be cleared.
16–18	—	Reserved. Should be cleared.
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.

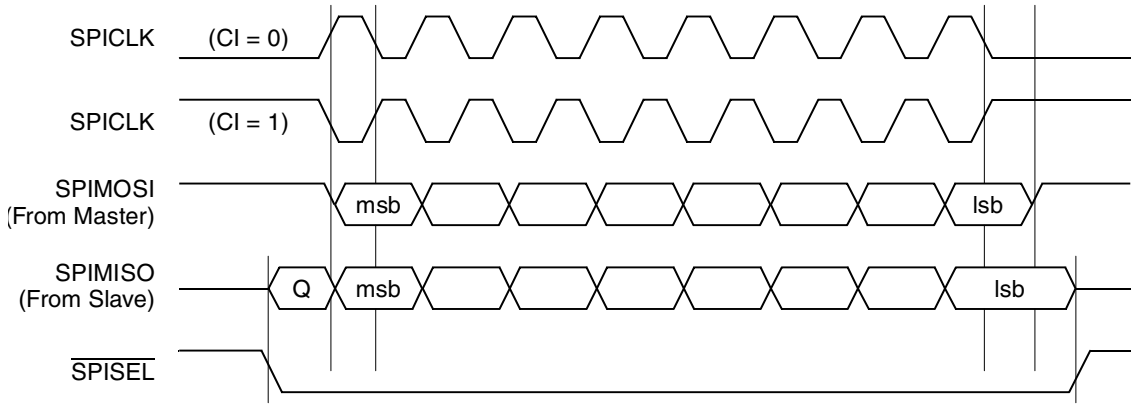
Figure 19-5 shows the SPI transfer format in which SPICLK starts toggling in the middle of the transfer (SPMODE[CP] = 0).



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

19.4.1.2 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.

Offset 0x024 Access: Mixed

	0	16	17	18	19	20	21	22	23	24	31
R	—		LT	DNR	OV	UN	MME	NE	NF	—	
W			w1c	w1c	w1c	w1c	w1c				
Reset	All zeros										

Figure 19-7. SPIE—SPI Event Register

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	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.

19.4.1.3 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

	0				16	17	18	19	20	21	22	23	24		31
R W	—					LT	DNR	OV	UN	MME	NE	NF	—		
Reset	All zeros														

Figure 19-8. SPIM—SPI Mask Register

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 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.

19.4.1.4 SPI Command Register (SPCOM)

The SPI command register (SPCOM), shown in Figure 19-9, is used to end SPI operation.

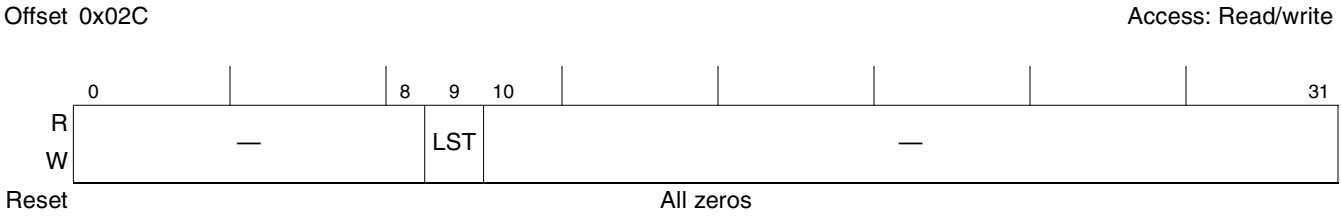


Figure 19-9. SPI Command Register (SPCOM)

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.

19.4.1.5 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.

Figure 19-10. SPI Transmit Data Hold Register

Table 19-8 shows the field descriptions of the SPI transmit data hold register.

Table 19-8. SPI Transmit Data Hold Register

Bits	Name	Description
0–31	DATA	These bits are the data to be sent.

19.4.1.6 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.

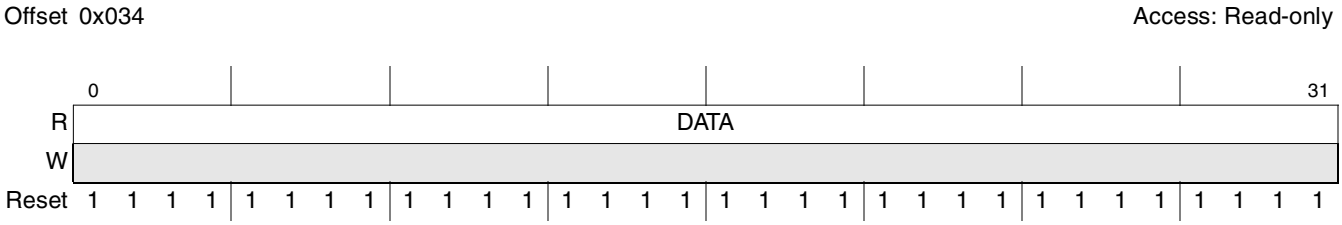


Figure 19-11. SPI Receive Data Hold Register

Table 19-9 shows the field descriptions of the SPI receive data hold register.

Table 19-9. SPI Receive Data Hold Register

Bits	Name	Description
0–31	DATA	Received data. These bits are the received data from the SPI bus.

19.4.1.6.1 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 bit (SPMODE[LEN] = 7).

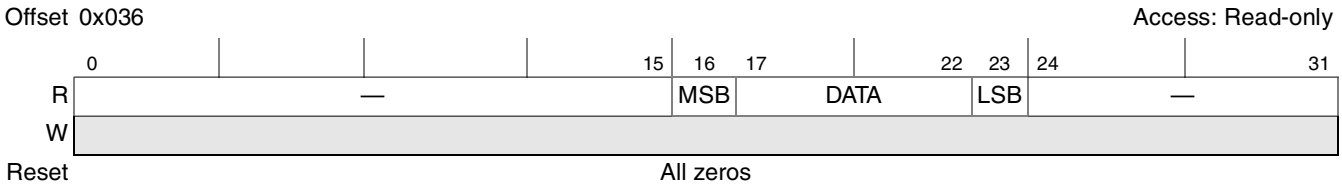


Figure 19-12. Example SPMODE[REV] = 0 SPMODE[LEN] = 7 LSB Sent First

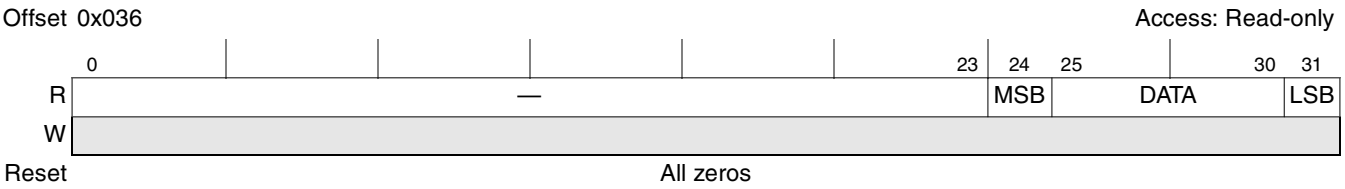


Figure 19-13. Example SPMODE[REV] = 1 SPMODE[LEN] = 7 MSB Sent First

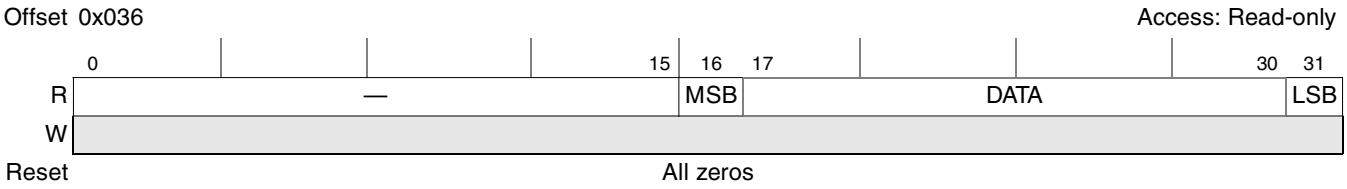


Figure 19-14. Example SPMODE[REV] = 1 SPMODE[LEN] = 15 MSB Sent First

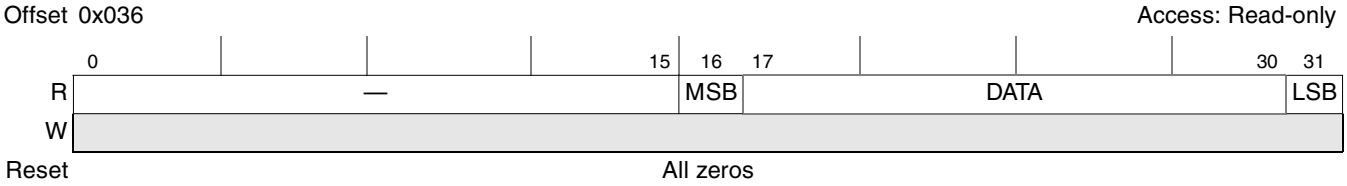


Figure 19-15. Example SPMODE[REV] = 0 SPMODE[LEN] = 15 LSB Sent First

19.5 Initialization/Application Information

The following sections describe programming examples of the SPI master and slave.

19.5.1 SPI Master Programming Example

The following sequence initialize 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 0xFFFFFFFF 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.

19.5.2 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 0xFFFFFFFF 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 MPC8349EA 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.

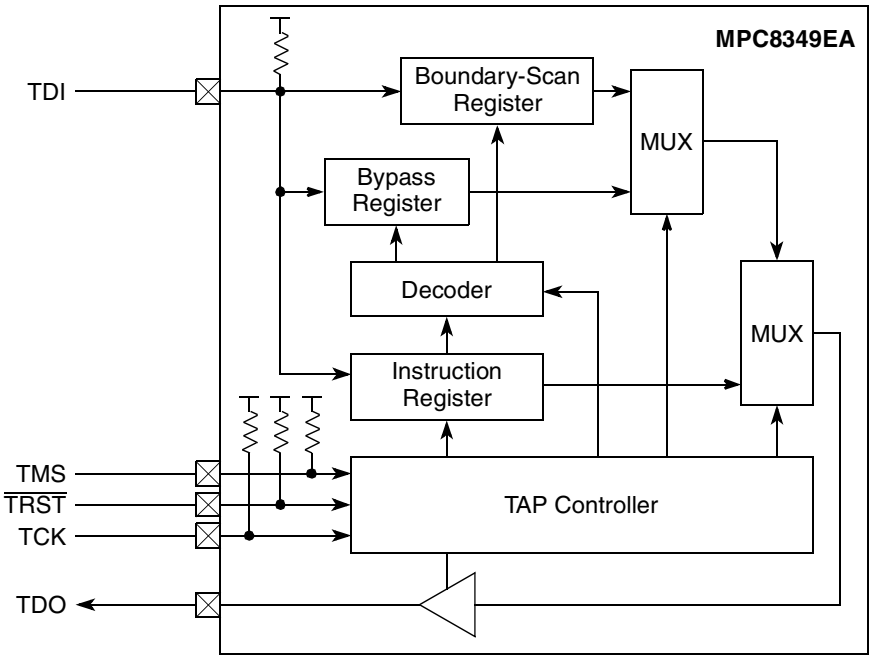


Figure 20-1. JTAG Interface Block Diagram

20.2 JTAG Signals

The MPC8349EA 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. Internally pulled up.	1	I
TDI	Test data input		Serial input for instructions and data to the JTAG test subsystem. Internally pulled up.	1	I
TDO	Test data output		Serial data output for the JTAG test subsystem. High impedance except when scanning out data.	Hi Impedance	O
TMS	Test mode select		Carries commands to the TAP controller for boundary scan operations. Internally pulled up.	1	I
$\overline{\text{TRST}}$	Test reset		Resets the TAP controller asynchronously.	—	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.	
		State Meaning	Asserted/Negated—Should be driven by a free-running clock signal with a 30–70% duty cycle. Input signals to the TAP are clocked in on the rising edge. Changes to the TAP output signals occur on the falling edge. The test logic allows TCK to be stopped. An unterminated input appears as a high signal level to the test logic due to an internal pull-up resistor.
		Timing	See IEEE 1149.1 standard for more details.
TDI	I	JTAG test data input.	
		State Meaning	Asserted/Negated—The value present on the rising edge of TCK is clocked into the selected JTAG test instruction or data register. An unterminated input appears as a high signal level to the test logic due to an internal pull-up resistor.
		Timing	See IEEE 1149.1 standard for more details.
TDO	O	JTAG test data output.	
		State Meaning	Asserted/Negated—The contents of the selected internal instruction or data register are shifted out on this signal on the falling edge of TCK. Remains in a high-impedance state except when scanning data.
		Timing	See IEEE 1149.1 standard for more details.

Table 20-2. JTAG Test—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
TMS	I	JTAG test mode select.	
		State Meaning	Asserted/Negated—Decoded by the internal JTAG TAP controller to distinguish the primary operation of the test support circuitry. An unterminated input appears as a high signal level to the test logic due to an internal pull-up resistor.
		Timing	See IEEE 1149.1 standard for more details.
$\overline{\text{TRST}}$	I	JTAG test reset.	
		State Meaning	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.
		Timing	See IEEE 1149.1 standard 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 compliance with the IEEE 1149.1 specification.

- 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 MPC8349EA. 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)

This chapter describes theory of operation of the general-purpose I/O (GPIO) modules, including a definition of the external signals and functions they serve. Additionally, the configuration, control, and status registers are described. Note that chapters in this reference manual describe additional specific initialization aspects for each individual block.

21.1 Introduction

This chapter describes the general-purpose I/O module, including pin descriptions, register settings and interrupt capabilities. The MPC8349EA includes two identical GPIO modules. Figure 21-1 shows the block diagram of one GPIO module.

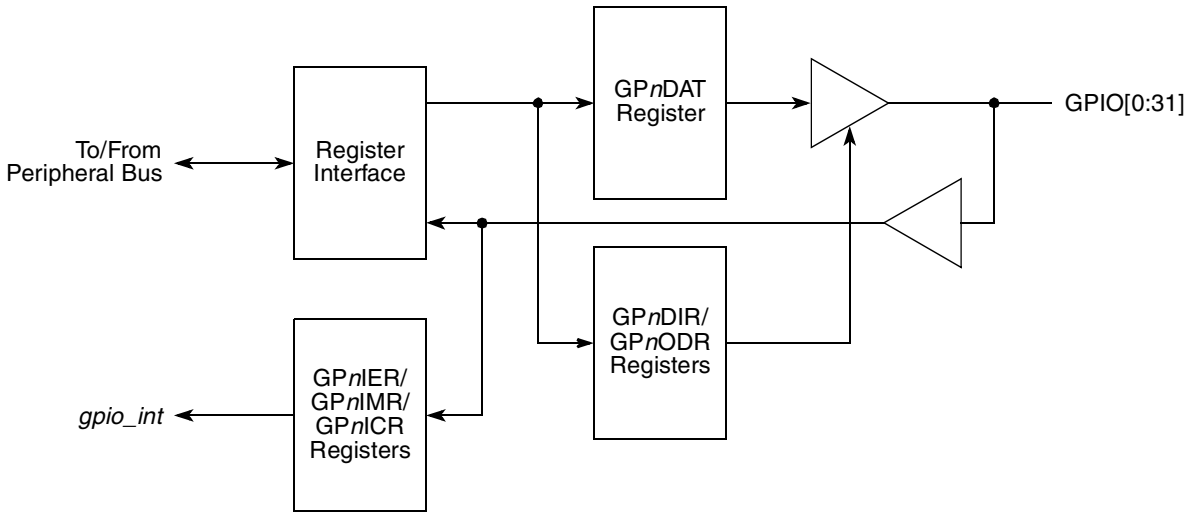


Figure 21-1. GPIO*n* Module Block Diagram

21.1.1 Overview

Each GPIO module supports 32 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 upon 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:

- 32 input/output ports
- Some ports have dedicated processor signals. Others are multiplexed together with other functional signals. See Chapter 3, “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 subsection 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 External Signals—Detailed Signal Descriptions

Signal	I/O	Description	
GPIO1[0:31] GPIO2[0:31]	I/O	General purpose I/O. Each signal can be set individually to act as input or output, according to application needs.	
		State Meaning	Asserted/Negated—Defined per application.
		Timing	Assertion/Negation—Inputs can be asserted completely asynchronously. Outputs are asynchronous to any externally visible clock

21.3 Memory Map/Register Definition

Each 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 GPIO starting address as defined in Chapter 2, “Memory Map.”

Table 21-2 shows memory map of GPIO.

Table 21-2. GPIO Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
0xC00	GPIO1 direction register (GP1DIR)	R/W	0x0000_0000	21.3.1/21-3
0xC04	GPIO1 open drain register (GP1ODR)	R/W	0x0000_0000	21.3.2/21-4
0xC08	GPIO1 data register (GP1DAT)	R/W	0x0000_0000	21.3.3/21-4

Table 21-2. GPIO Register Address Map (continued)

Offset	Register	Access	Reset Value	Section/ Page
0xC0C	GPIO1 interrupt event register (GP1IER)	w1c	Undefined	21.3.4/21-5
0xC10	GPIO1 interrupt mask register (GP1IMR)	R/W	0x0000_0000	21.3.5/21-5
0xC14	GPIO1 external interrupt control register (GP1ICR)	R/W	0x0000_0000	21.3.6/21-6
0xC1C–0xCFF	Reserved	—	—	—
0xD00	GPIO2 direction register (GP2DIR)	R/W	0x0000_0000	21.3.1/21-3
0xD04	GPIO2 open drain register (GP2ODR)	R/W	0x0000_0000	21.3.2/21-4
0xD08	GPIO2 data register (GP2DAT)	R/W	0x0000_0000	21.3.3/21-4
0xD0C	GPIO2 interrupt event register (GP2IER)	R/W	Undefined	21.3.4/21-5
0xD10	GPIO2 interrupt mask register (GP2IMR)	R/W	0x0000_0000	21.3.5/21-5
0xD14	GPIO2 external interrupt control register (GP2ICR)	R/W	0x0000_0000	21.3.6/21-6
0xD1C–0xDFF	Reserved	—	—	—

21.3.1 GPIO*n* Direction Register (GP1DIR–GP2DIR)

The GPIO*n* direction registers (GP1DIR–GP2DIR), shown in Figure 21-2, defines the direction of the individual ports.

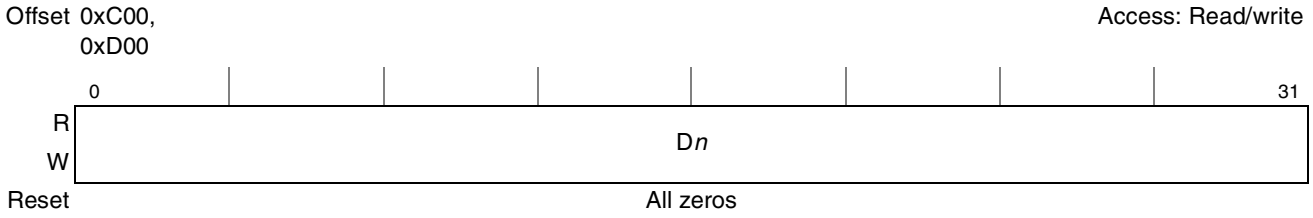


Figure 21-2. GPIO*n* Direction Register (GP*n*DIR)

Table 21-3 defines the bit fields of GP*n*DIR.

Table 21-3. GP*n*DIR Bit Settings

Bits	Name	Description
0–31	D*n*	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.

21.3.2 GPIO*n* Open Drain Register (GP1ODR–GP2ODR)

The GPIO*n* open drain register (GP1ODR–GP2ODR), shown in Figure 21-3, defines the way individual ports drive their output.

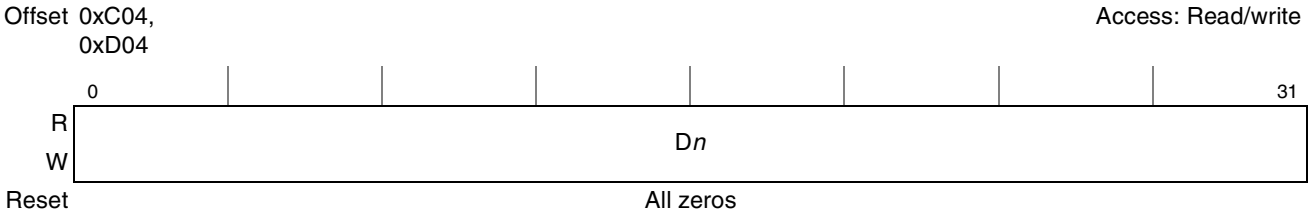


Figure 21-3. GPIO*n* Open Drain Register (GP1ODR–GP2ODR)

Table 21-4 defines the bit fields of GP*n*ODR.

Table 21-4. GP*n*ODR Bit Settings

Bits	Name	Description
0–31	D*n*	Open-drain configuration. Indicates whether a signal is actively driven as an output or an open-drain driver. This register has no effect on signals programmed as inputs in the corresponding GP*n*DIR. 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 tri-stated.

21.3.3 GPIO*n* Data Register (GP1DAT–GP2DAT)

The GPIO*n* data register (GP1DAT–GP2DAT), shown in Figure 21-4, carries the data in/out for the individual ports.

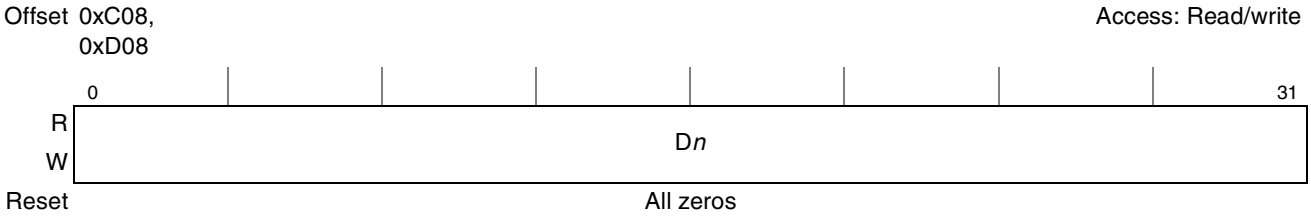


Figure 21-4. GPIO*n* Data Register (GP1DAT–GP2DAT)

Table 21-5 defines the bit fields of GP*n*DAT.

Table 21-5. GP*n*DAT Bit Settings

Bits	Name	Description
0–31	D*n*	Data. Write data is latched and presented on external signals if GP*n*DIR has configured the port as an output. Read operation always returns the data at the signal.

21.3.4 GPIO*n* Interrupt Event Register (GP1IER–GP2IER)

The GPIO*n* interrupt event register (GP1IER–GP2IER), shown in Figure 21-5, carries information of the events that caused an interrupt. Each bit in GP*n*IER, corresponds to an interrupt source. GP*n*IER bits are cleared by writing ones. However, writing zero has no effect.

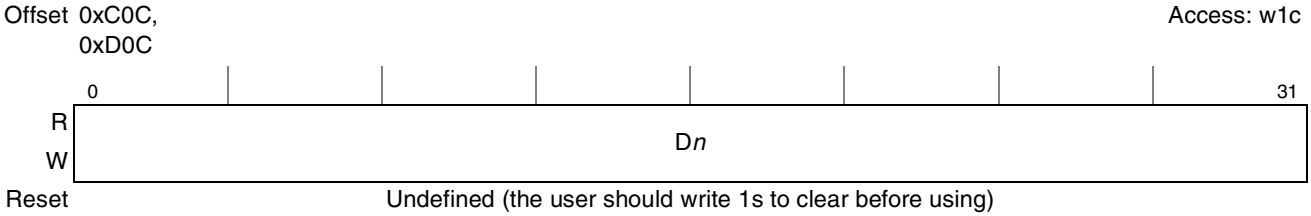


Figure 21-5. GPIO*n* Interrupt Event Register (GP1IER–GP2IER)

Table 21-6 defines the bit fields of GP*n*IER.

Table 21-6. GP*n*IER Bit Settings

Bits	Name	Description
0–31	D*n*	Interrupt events. Indicates whether 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.

21.3.5 GPIO*n* Interrupt Mask Register (GP1IMR–GP2IMR)

The GPIO*n* interrupt mask register (GP1IMR–GP2IMR), shown in Figure 21-6, defines the interrupt masking for the individual ports. When a masked interrupt request occurs, the corresponding GP*n*IER bit is set, regardless of the GP*n*IMR state. When one or more non-masked interrupt events occur, the GPIO module issues an interrupt to the on chip interrupt controller.

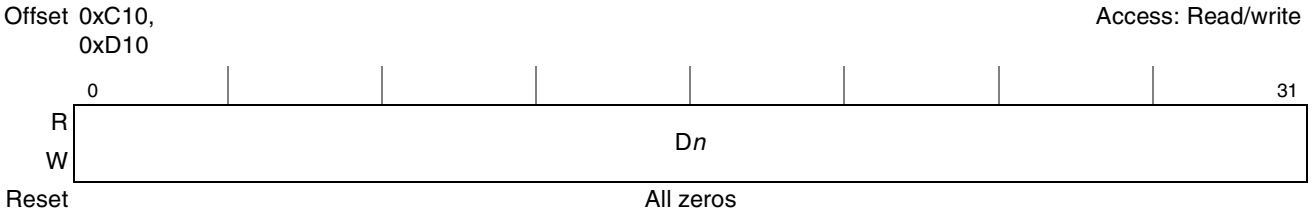


Figure 21-6. GPIO*n* Interrupt Mask Register (GP1IMR–GP2IMR)

Table 21-7 defines the bit fields of GP*n*IMR.

Table 21-7. GP*n*IMR Bit Settings

Bits	Name	Description
0–31	D*n*	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).

21.3.6 GPIO*n* Interrupt Control Register (GP1ICR–GP2ICR)

The GPIO*n* interrupt control register (GP1ICR–GP2ICR), shown in Figure 21-7, determines whether the corresponding port line asserts an interrupt request upon either a high-to-low change or any change on the state of the signal.

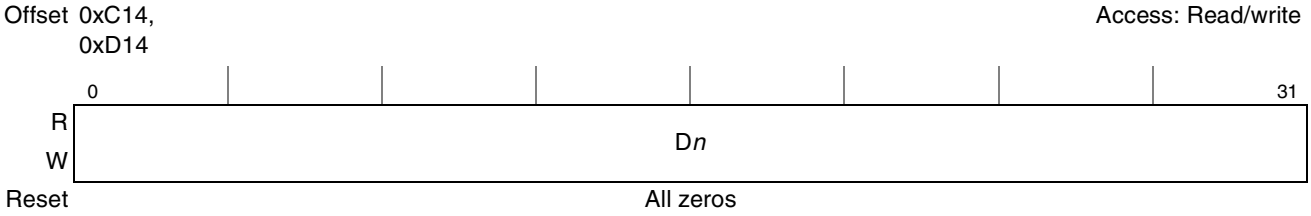


Figure 21-7. GPIO*n* Interrupt Control Register (GP1ICR–GP2ICR)

Table 21-8 defines the bit fields of GP*n*ICR.

Table 21-8. GP*n*ICR Bit Settings

Bits	Name	Description
0–31	D*n*	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.

Chapter 22
Delay Lock Loop (DLL)

This chapter describes the theory of operation of the delay lock loop (DLL) module in the integrated device. Additionally, the configuration, control, and status registers are described. Note that other chapters in this book describe additional specific initialization aspects for individual blocks.

22.1 Introduction

The DLL unit consists of a phase detection circuit, adjustable delay unit, register file and control logic. A high-level block diagram of these elements is shown in Figure 22-1.

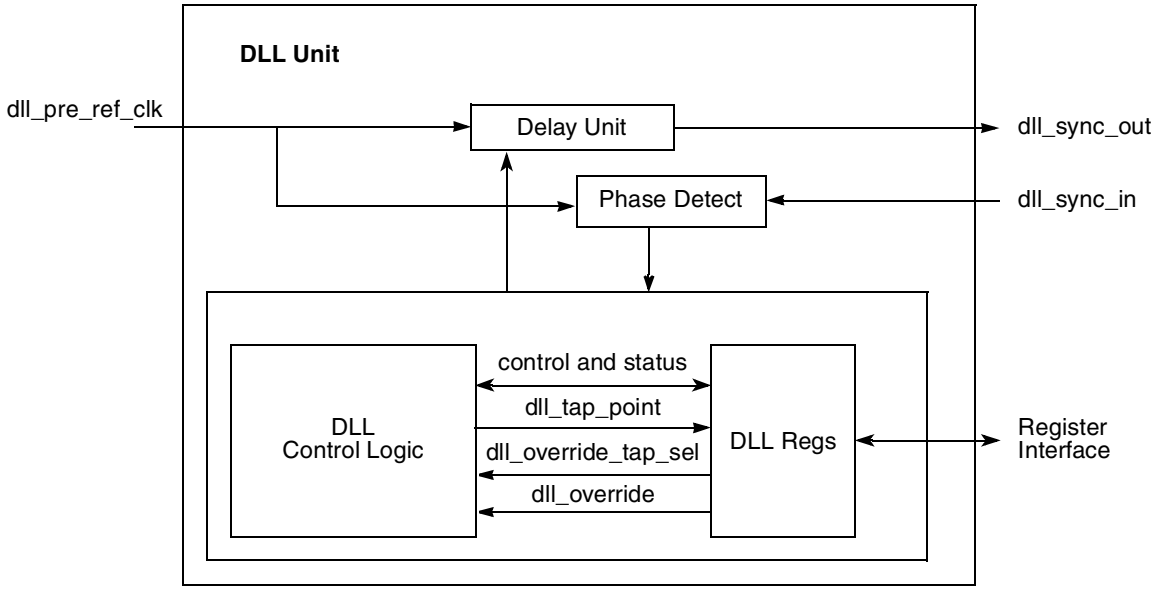


Figure 22-1. DLL Block Diagram

22.2 Overview

The purpose of the DLL macro is to shift a reference clock to create an output clock to be used by external memory or I/O devices. The output clock is created by propagating the reference clock through a delay chain. A phase detect circuit compares the phase of the reference clock with that of a feedback clock. An output TAP point of the delay chain is chosen for the output clock such that the feedback clock is phase aligned with the reference clock. The DLL macro also contains override, debug, and error features.

DLL is used for LCLK (local bus clocks) skew elimination. Therefore, DLL_SYNC_OUT/DLL_SYNC_IN should be interpreted as LSYNC_OUT/LSYNC_IN, and DLL_CLK_OUT[0:n] should be interpreted as LCLK[0:n].

22.2.1 Features

- LOCK status reports when phase lock search was completed.
- WRAP status reports when the DLL wraps beyond either end of its delay chain during the search process. If WRAP is set while LOCK is set, this indicates a failure to find a lock during the search process.
- OVERRIDE mode allows a specific tap point to be selected.
- Current tap point is continuously reported.

22.2.2 Modes of Operation

- Normal mode: The DLL can operate in normal mode, in which the phase detector circuitry dynamically adjusts the tap point of the delay line.
- Override mode: The DLL can operate in override mode in which the tap point is specified by an input vector.

22.2.3 External Signals

Table 22-1 displays the signals of the DLL macro.

Table 22-1. DLL Macro External Signals

Signal	I/O	Signal Description
DLL_SYNC_OUT	O	DLL output clock. Phase aligned with other clock outputs (DLL_CLK_OUT[0:n]).
DLL_SYNC_IN	O	DLL feedback clock, phased aligned with the internal logic clock.
DLL_CLK_OUT[0:n]	O	DLL output clocks to be used by external on-board devices.

22.3 Initialization and Application Information

An example application using the DLL macro is shown in Figure 22-2. As this example shows, a memory or I/O controller generates the reference clock based on an internal knowledge of the ratio between the external clock and internal clk. DLL_CLK_OUT goes to the external memory or I/O devices, and the DLL_SYNC_OUT must be connected back to the DLL_SYNC_IN while keeping the same trace length as of DLL_CLK_OUT path. The DLL macro uses the phase detection and the delay chain to align the internal logic clock with the external device input clock, such that the integrated device's flip-flops and the external device's flip-flops are fed by clocks with minimal skew between them.

DLL lock status is used by the memory controller to hold off memory access until DLL search has been completed. However, if DLL override mode is used, DLL lock has no meaning.

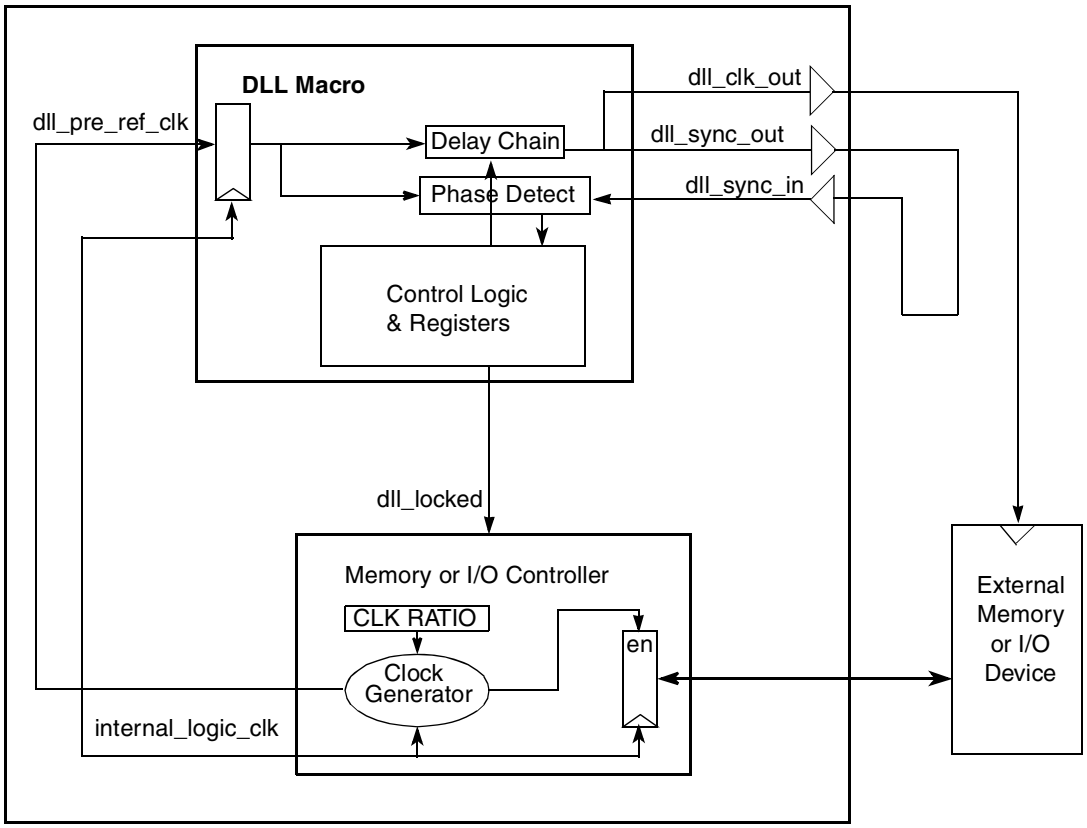


Figure 22-2. DLL Application Example

22.4 Memory Map/Register Definition

The DLL programmable register map occupies 20 bytes of memory-mapped space. Reading undefined portions of the memory map returns all zeros; writing has no effect.

All DLL registers are 32 bits wide located on 32-bit address boundaries. All addresses used in this chapter are offsets from the DLL starting address as defined in Chapter 2, "Memory Map."

Table 22-2 shows the DLL registers.

Table 22-2. DLL Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
0x00	Reserved. Reset value should be preserved.	R/W	0x0500_0280	—
0x04	Reserved. Reset value should be preserved.	R/W	0x8004_0810	—
0x08	DLL override register (DLLOVR)	R/W	0x0000_0000	22.4.1/22-4
0x0C	DLL status register (DLLSR)	R	0x0000_0000	22.4.2/22-4
0x10	DLL clock register (DLLCK)	R/W	0xFC00_0000	22.4.3/22-5
0x14–0xFF	Reserved.	—	—	—

22.4.1 DLL Override Register (DLLOVR)

The DLL override register (DLLOVR), shown in Figure 22-3, controls the override operation and the override value (coarse or fine) to force on the DLL in override mode.

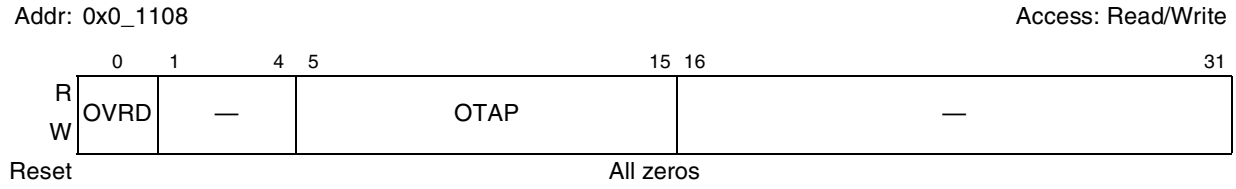


Figure 22-3. DLL Override Register

DLLOVR can be used to force a fixed delay value. It is recommended to wait until DLL lock is achieved, read the measured course and TAP delay values from the DLL status register, and then set the override delay values accordingly. Table 22-3 describes DLLOVR fields.

Table 22-3. DLLOVR Field Description

Bits	Name	Description
0	OVRD	DLL override mode. 0 Normal mode 1 Override mode
1–4	—	Reserved
5–15	OTAP	DLL override coarse and TAP select. OTAP[5–7] Coarse delay select OTAP[8–15] TAP delay select
16–31	—	Reserved

22.4.2 DLL Status Register (DLLSR)

DLLSR shown in Figure 22-4, indicates the DLL status. It is a read only register.

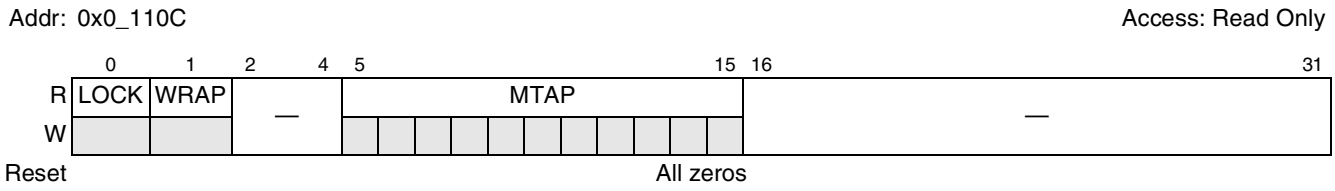


Figure 22-4. DLL Status Register

Table 22-4 describes DLLSR fields.

Table 22-4. DLLSR Field Description

Bits	Name	Description
0	LOCK	DLL locked. When the DLL finished its TAP point search, the LOCK bit is set. The LOCK bit must be considered together with the WRAP bit state.
1	WRAP	DLL wrapped. When the DLL LOCK is set, WRAP provides information for the DLL condition. If WRAP = 1, it indicates that the DLL search for a TAP point has completed unsuccessfully. When the DLL search was successful, the WRAP state remains clear while LOCK is set.
2–4	—	Reserved
5–15	MTAP	Measured coarse and tap delay by the DLL. MTAP[5–7] Coarse delay MTAP[8–15] Tap delay
16–31	—	Reserved

22.4.3 DLL Clock Register (DLLCK)

DLLCK shown in Figure 22-5, enables or disables the signals clock out.

Addr: 0x0_1110 Access: Read/Write

	0	1	2	3																										31
R W	CE0	CE1	CE2	—																										
Reset	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-5. DLL Clock Register

Table 22-5 describes DLLCK fields.

Table 22-5. DLLCK Field Description

Bits	Name	Description
0	CE0	Enable/Disable LCLK[0] signal clock out 0 Disable LCLK[0] 1 Enable LCLK[0]
1	CE1	Enable/Disable LCLK[1] signal clock out 0 Disable LCLK[1] 1 Enable LCLK[1]
2	CE2	Enable/Disable LCLK[2] signal clock out 0 Disable LCLK[2] 1 Enable LCLK[2]
3–31	—	Reserved

Appendix A
Revision History

This appendix provides a list of the major differences between revision 0 and revision 1 of the ***MPC8349EA PowerQUICC II Pro Integrated Host Processor Family Reference Manual***.

Section/Page	Changes
Cover and Title page	The manual cover and title page should say: Supports: MPC8349EA MPC8347EA MPC8343EA
1.1, 1-1	In Table 1-1, the number of GPIO signals should be: MPC8347E—52 MPC8343E—39
2.1, 2-1	Add this section immediately after Section 2.1:

2.2 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.

2.2, 2-1	Before Table 2-1, add the following paragraphs: Reading from address locations which appear as reserved in the memory map table is not guaranteed to return predictable data. Writing to address locations which 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 will be 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.
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 will indicate when this is needed.

2.2, 2-19 — In Table 2-2, change the address of register SI_CTRL from 0x2_340C to 0x2_3410 and the address of PRI_CTRL from 0x2_3410 to 0x2_340C.

3.1, 3-2 — In Figure 3-1 (page 3-2) and in Table 3-1 (page 3-5), change CPCI1_HS_LED to CPCI_HS_LED and CPCI1_HS_ENUM to CPCI_HS_ENUM.

3.1, 3-4 — In both Table 3-1 and Table 3-2 (page 3-13), the signals MA and MBA have been relabeled. Thus, signals listed previously as MA[0:14] and MBA[0:2] should be changed to MA[14:0] and MBA[2:0]. Note that the sense of these signals is unchanged—MA0 and MBA0 are still the lowest-ordered signals.

3.1, 3-6 — In both Table 3-1 and Table 3-2 (page 3-16), add the following footnote to pin TSEC1_TXD3.

Table 3-1. MPC8349E Signal Reference by Functional Block

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
TSEC1_TXD3[1]	TSEC1 transmit data 3	TSEC1	1	I/O	15-1/15-9	—	4-1/4-1

[1] This pin should be pulled high during a hard reset for proper functionality of the device since it has a weak internal pull up. No external pull-down resistors are allowed to be mounted on this net.

4.4.2.9.1, 4-18 — In Table 4-18, change CPCI1_HS_LED to CPCI_HS_LED, CPCI1_HS_ES to CPCI_HS_ES, and CPCI1_HS_ENUM to CPCI_HS_ENUM.

4.5.3, 4-33 — Change the first sentence to:

As shown in Figure 4-9, the primary clock input (PCI_CLK/PCI_SYNC_IN) frequency is multiplied up by the system phase-locked loop (PLL) and the clock unit to create three major clock domains:

4.6.1.5, 4-38 — In Figure 4-12, change the field name for bits 16–31 for “RPRW” to “RCPW.”

5.2.4.1.1, 5-6 — Add the following before the second paragraph:

- 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**.

5.2.4.1.1, 5-6 Replace Table 5-6 with the following:

Table 5-6. ALTCBAR 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.

5.3.2.5, 5-25 In Table 5-27, bits 7–10 and bits 14–17 should have reference to notes ‘1’ and ‘2’ in superscript. Also, bits 16 and 17 are swapped. Configure SICRL[17] to control GPIO1[9] and configure SICRL[16] to control GPIO1[10]. Replace the following rows:

Table 5-27. SICRL Bit Settings

7[1]	GPIO1_B	GPIO1[1]	$\overline{\text{GTM1_TGATE1}}$/$\overline{\text{GTM2_TGATE2}}$	—	—
8[1]	GPIO1_C	GPIO1[2]	$\overline{\text{GTM1_TOUT1}}$	—	—
9[1]	GPIO1_D	GPIO1[3]	GTM1_TIN2/GTM2_TIN1	—	—
10[1]	GPIO1_E	GPIO1[4]	$\overline{\text{GTM1_TGATE2}}$/$\overline{\text{GTM2_TGATE1}}$	—	—
11–12	GPIO1_F	GPIO1[5]	$\overline{\text{GTM1_TOUT2}}$	$\overline{\text{GTM2_TOUT1}}$	$\overline{\text{DMA_DDONE1}}$
13[2]	GPIO1_G	GPIO1[6]	GTM1_TIN3/GTM2_TIN4	—	—
14[2]	GPIO1_H	GPIO1[7]	$\overline{\text{GTM1_TGATE3}}$/$\overline{\text{GTM2_TGATE4}}$	—	—
15[2]	GPIO1_I	GPIO1[8]	$\overline{\text{GTM1_TOUT3}}$	—	—
16[2]	GPIO1_J	GPIO1[10]	$\overline{\text{GTM1_TGATE4}}$/$\overline{\text{GTM2_TGATE3}}$	—	—
17[2]	GPIO1_K	GPIO1[9]	GTM1_TIN4/GTM2_TIN3	—	—

5.3.2.6, 5-29 In Table 5-28, the reset value of bit 27 should be 0. Also, add a footnote to PCI. Replace bit 27 row and add note 1 at the end of the table as follows:

Table 5-28. SICRH Bit Settings

27	PCI[1]	PCI_CLK_OUT[3:4]	$\overline{\text{LCS}}$[6:7]	—	0

1 PCI_CLK[3:4]/$\overline{\text{LCS}}$[6:7] drive zeros during and after the period in which $\overline{\text{PORESET}}$ and $\overline{\text{HRESET}}$ are asserted. If a slave device is connected to PCI_CLK[3]/$\overline{\text{LCS}}$[6] or PCI_CLK[4]/$\overline{\text{LCS}}$[7] and this slave will drive data on LAD signals when it has $\overline{\text{LCS}}$ and $\overline{\text{LOE}}$ both asserted, this will lead to contention on the LAD bus and errors in loading the RCW data. It will also lead to errors when the e300 CPU will try to fetch boot data.
The user should take one of the following precautions:
Use an I^2C EEPROM to load RCW, and use the boot sequencer to set SICRH[27] before letting the CPU boot its code.
If possible, use a hard coded RCW and use the remote PCI host to set SICRH[27] before letting the CPU boot its code.
If possible, use LCS[6:7] multiplexed with SPI pins. See Section 5.3.2.5, “System I/O Configuration Register Low (SICRL).”
Do not connect local bus slaves to LCS[6:7].

5.4.4.1, 5-36 — In Table 5-33, replace the description of bit 31 with the following:

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.

5.5.6.1, 5-47 — In the first paragraph (fifth sentence) replace the sentence, ‘The RTC can be clocked by the internal system bus clock or by an external 32.768 kHz crystal.’ with the following:

The RTC can be clocked by the internal system bus clock or an external clock source.

5.6.5, 5-39 — In Table 5-40, change the reset value of RTALR to 0xFFFF_FFFF.

6.2.3, 6-5 — In Figure 6-3, change the access to ‘User w1c.’

Table 6-7. AEATR Field Descriptions

Bits	Name	Description
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

7.1, 7-3 — Remove references to a moded 32- or 64-bit data bus; it is only a 64-bit data bus. Change the first sentence of the third paragraph to:

The e300 has a 64-bit data bus and a 32-bit address bus.

7.1.1, 7-4 — Change the first sub-bullet under ‘Facilities for enhanced system performace’ to:

A 64-bit split-transaction internal data bus interface to the coherent system bus (CSB) with burst transfers.

7.3.7, 7-36 — Change the second sentence of the third paragraph to:

The interface includes a 32-bit address bus, a 64-bit data bus, and 56 control and information signals (see Figure 7-4).

7.3.7.1, 7-37 — The CSB bus is not selectable between 32-bit and 64-bit since it is only 64-bit wide. Replace the text of this section with the following:

The e300 core 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.

8.5.2, 8-10 — In Table 8-6, replace the listed internal ‘Interrupt Meaning’ row for with the following:

ipp_ind_ext_int[0]	with	$\overline{\text{IRQ0}}$
ipp_ind_ext_int[1]	with	$\overline{\text{IRQ1}}$
ipp_ind_ext_int[2]	with	$\overline{\text{IRQ2}}$
ipp_ind_ext_int[3]	with	$\overline{\text{IRQ3}}$
ipi_int_internal[32]	with	RTC SEC
ipi_int_internal[33]	with	PIT
ipi_int_internal[34]	with	PCI1
ipi_int_internal[35]	with	PCI2

8.5.19, 8-26 — In Figure 8-23, change the offset to 0x60.

8.5.20, 8-27 — In Figure 8-24, change the offset to 0x64.

Chapter 9 — The signals MDQS[0:8] should be removed. Note this only applies to the active-low signal, and the signal MDQS[0:8] still remains. Instances where MDQS[0:8] appear:

Figure 9-1 (page 9-2), ‘DDR Memory Controller Simplified Block Diagram’ (in the ‘Data Qualifiers’ signals group).

Table 9-1 (page 9-4), ‘DDR Memory Interface Signal Summary’ (the entire signal name should be removed).

Table 9-3 (page 9-5), ‘Memory Interface Signals—Detailed Signal Descriptions’ (the signal name should be removed). The line in the description of the I/O function, ‘The data strobes may be single ended or differential.’ should also be removed. The table row should read:

Table 9-3. Memory Interface Signals—Detailed Signal Descriptions

MDQS[0:8]	I/O	Data strobes. Inputs with read data and outputs with write data.

In Figure 9-33 (page 9-39), ‘DDR Memory Controller Block Diagram’ (in the data strobe signals group).

9.3.2.1, 9-8 — In Table 9-3, the MDIC[0:1] description should read:

Driver impedance calibration. MDIC0 is grounded through an 18.2-Ω precision 1% resistor and MDIC1 is connected to GVDD through an 18.2-Ω precision 1% resistor. ...

9.4, 9-10 — In Table 9-5, remove the registers DDR_INIT_EXT_ADDR and CAPTURE_EXT_ADDR since extended address is not supported.

9.4.1.1, 9-10 — In Figure 9-2, bits 4-7 and 20-23 should be shown as reserved. SA*n* should be bits 8-15 and EA*n* should be bits 24-31. Replace the figure with the following:

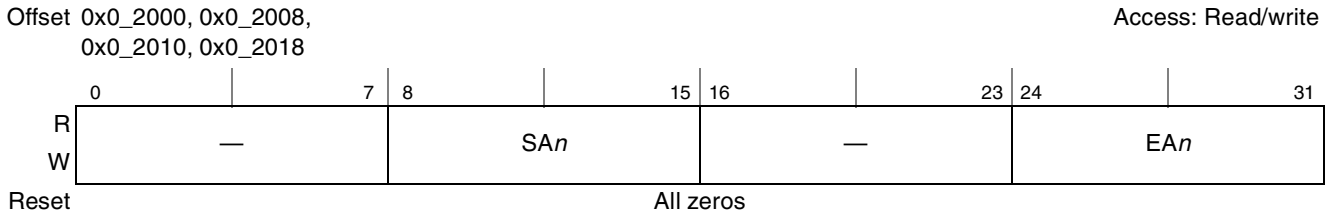


Figure 9-2. Chip Select Bounds Registers (CS*n*_BNDS)

9.4.1.1, 9-11 In Table 9-6, replace the table with the following:

Table 9-6. CS*n*_BNDS Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8–15	SA*n*	Starting address for chip select (bank) *n.* This value is compared against the 8 msbs of the 32-bit address.
16–23	—	Reserved
24–31	EA*n*	Ending address for chip select (bank) *n.* This value is compared against the 8 msbs of the 32-bit address.

9.4.1.14, 9-27 DDR_SDRAM_CLK_CNTL[SS_EN] has been removed from the MPC8349EA and thus there is no effect of setting or clearing this bit. The device will work in source sync mode by default.

9.4.1.15, 9-28 Remove references to DDR_INIT_ADDR data skew adjustment since this feature is not supported. Remove the NOTE and change the section text to:

The DDR SDRAM initialization address register, shown in Figure 7-18, provides the address that will be used for the automatic CAS to preamble calibration after POR.

9.4.1.16, 9-29 Remove this section, since extended address is not supported.

9.4.1.30, 9-37 Remove this section, since extended address is not supported.

9.5.2, 9-45 The DDR address multiplexing tables, Tables 9-40 through 9-43, should be labeled for 32-bit addressing instead of 36-bit addressing. The column headings should be changed to read:

Table 9-40. DDR1 Address Multiplexing for 64-bit Data Bus with Interleaving Disabled

Row x 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	29–31

9.6-9-65 In Table 9-52, remove DDR_INIT_EXT_ADDR since extended address is not supported.

10.2, 10-5 In Table 10-1, change the signal LCLK[0:2] row to read:

Table 10-1. Signal Properties—Summary

LCLK[0:2]	—	—	Local bus clocks. Enabled by the DLLCK register. See Section 22.4.3, “DLL Clock Register (DLLCK),” for details.	3	O	Driven

10.3.1.2.1, 10-12	In the first paragraph, replace the first two sentences with the following: The address mask fields of the option registers (OR*n*[AM]) mask up to 17 corresponding BR*n*[BA] fields. The 15 lsbs of the 32-bit internal address do not participate in bank address matching in selecting a bank for access.”
10.3.1.2.1, 10-13	In Table 10-5, the first column, ‘Bits 17-18,’ does not apply, delete the entire column from the table.
10.4, 10-33	In Figure 10-20, change ‘34-bit System Address’ to ‘32-Bit System Address.’
10.4.1.1, 10-33	In the first paragraph, change the third sentence to the following: 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]).
10.4.2.4, 10-48	In Table 10-27, remove the XBA row.
10.4.3.11.1, 10-59	Replace ‘RFCR’ with ‘RFRC’ (Refresh Recovery) twice in the section and once in Figure 10-50.
10.5.4.3.3, 10-93	In Table 10-39, remove the XBA and XAM rows.
10.5.4.3.5, 10-94	In Table 10-43, remove the XBA and XAM rows.
11.4.3, 11-4	Remove the POCMR streaming enable (SE) bit from Figure 11-4 and Table 11-3 and change it to ‘Reserved’:

0x10, 0x28, 0x40, 0x58, 0x70, 0x88

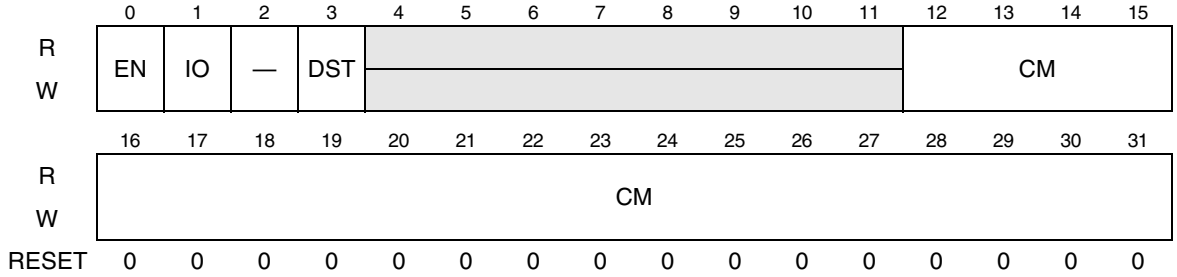


Figure 11-4. PCI Outbound Comparison Mask Registers (POCMR*n*)

Table 11-3. POCMR*n* Field Descriptions

Bits	Name	Description
2	—	Reserved

12.5.5.1, 12-8	In the second paragraph, replace the sentence with the following: Table 12-7 describes the ODR register.
Chapter 13, 13-1	Replace ‘compliant’ with ‘compatible.’
13.1.2, 13-3	Replace ‘compliant’ with ‘compatible.’
13.1.3.1, 13-4	Append the following sentence at the end of the second paragraph:

In agent mode, configuration cycles are acknowledged if CFG_LOCK is 0 (see Section 13.3.3.24, “PCI Function Configuration Register”), either from reset configuration or after being cleared by software.

13.2, 13-4 — In Table 13-2, Figure 13-2, and Table 13-3, replace signals CPCI1_HS_ENUM with CPCI_HS_ENUM, CPCI1_HS_ES with CPCI_HS_ES, and CPCI1_HS_LED with CPCI_HS_LED.

13.3.1, 13-15 — In the first paragraph, add the following sentence to the end of the paragraph:

The PCI registers, PCI_CONFIG_ADDRESS, PCI_CONFIG_DATA, and PCI_INT_ACK, are little endian registers.

13.3.3.6, 13-36 — The reset value of Figure 13-25 should read ‘All zeros’ instead of a blank.

13.3.3.7, 13-37 — The reset value of Figure 13-26 should read ‘0010_0000’ instead of a blank.

13.3.3.8, 13-37 — The reset value of Figure 13-27 should read ‘0000_1011’ instead of a blank.

13.3.3.26, 13-47 — In Table 13-43, replace signal CPCI1_HS_ENUM with CPCI_HS_ENUM in the description bit 17.

14.3, 14-11 — In Table 14-2, replace the following rows with the following:

Table 14-2. SEC Base Address Map

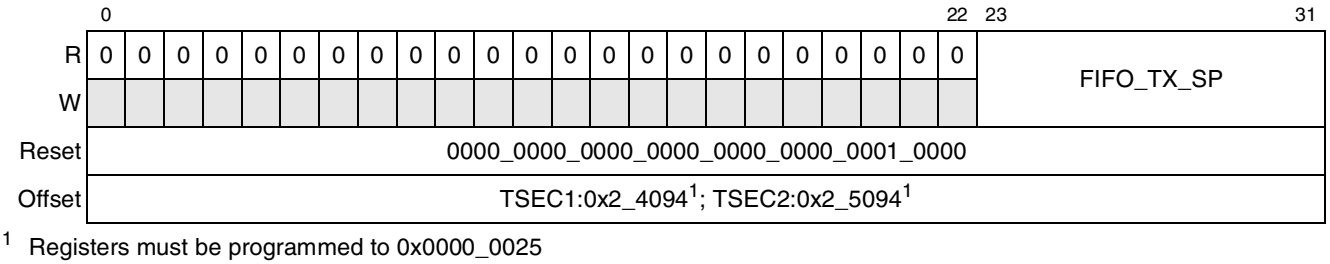
0x3_2000–0x3_2FFF	DEU	DES/3DES execution unit	Crypto EU	14.5.2/14-38
0x3_4000–0x3_4FFF	AESU	AES execution unit		14.5.6/14-74
0x3_6000–0x3_6FFF	MDEU	Message digest execution unit		14.5.4/14-55

14.4.5, 14-25 — In Table 14-9, replace the entire row for the descriptor type 1010_1 with the following:

Table 14-9. Descriptor Pointer Dword Usage

Descriptor Type	Pointer DWORD 1	Pointer DWORD 2	Pointer DWORD 3	Pointer DWORD 4	EXTENT 4	Pointer DWORD 5	EXTENT 5	Pointer DWORD 6	EXTENT 6	Pointer DWORD 7
1010_1	nil	nil	nil	In 1	undefined	In 2	undefined	In 3	undefined	Out

15.5.3.2.2, 15-30 — In Figure 15-15, the footnote of FIFO_TX_SP register applies to both TSEC1 and TSEC2. Add the footnote to the offset for TSEC2. Replace the figure with the following:



[1] Registers must be programmed to 0x0000_0025

Figure 15-15. FIFO_TX_SP Register Definition

15.7.1.1, 15-126 In Table 15-123, IEVENT should be written to with 1s, instead of 0s, to be cleared, replace the row with the following:

Table 15-123. MII Mode Register Initialization Steps

Clear IEVENT register, IEVENT[1111_1111_1111_1111_1111_1111_1111_1111]	

16.3, 16-17 Table 16-7, the address offsets of PRI_CTRL and SI_CTRL registers were exchanged. The offsets should be reversed so that PRI_CTRL is 0x40C and SI_CTRL is 0x410. Replace the rows with the following:

Table 16-7. USB Interface Memory Map

0x2_2410	SI_CTRL—System interface control	R/W	0x0000_0000	16.3.2.28/16-64
0x2_240C	PRI_CTRL—Priority control	R/W	0x0000_0000	16.3.2.29/16-65

16.3, 16-18 In Table 16-7, replace the following row with:

Table 16-7. USB Interface Memory Map

Offset	Register	Access	Reset	Section/Page
0x2_3184	PORTSC1—Port status/control	R/W	0x8C00_0000	16.3.2.16/16-39

16.3, 16-20 The registers with offsets Base + 000h to + 14h on pages 16-20 and 16-21 appeared in the manual erroneously and should be removed.

16.3.2.2, 16-30 Add ULPII bit 10 to Figure 16-9 and change bits 15-12 and 10 to read-only. Also, change ‘Clear’ to ‘w1c.’

Base +144h

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0000_0000_0000_0000															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	AS	PS	RCL	HCH		ULPII		SLI	SRI	URI	AAI	SEI	FRI	PCI	UEI	UI
W								w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0000_0000_0000_0000															

Figure 16-9. USB Status Register (USBSTS)

16.3.2.3, 16-32 Add ULPIE bit 10 to Figure 16-10:

Base +148h

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0000_0000_0000_0000															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R						ULPIE		SLE	SRE	URE	AAE	SEE	FRE	PCE	UEE	UE
W																
Reset	0000_0000_0000_0000															

Figure 16-32. USB Interrupt Enable (USBINTR)

16.3.2.15, 16-44 Add CF bit 0 to Figure 16-21 and Table 16-28:

Base +180h

	31														1	0
R	—															CF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 16-21. Configure Flag Register (CONFIGFLAG)

Table 16-28. CONFIGFLAG Register Field Descriptions

Bits	Name	Description
31–1	—	Reserved.
0	CF	Configure flag. Always 1 indicating all port routings default to this host.

16.3.2.16, 16-45 Replace Figure 16-22 with:

Offset 0x2_2184 (MPH PORTSC1), 0x2_2188 (MPH PORTSC2)
0x2_3184 (DR PORTSC1)

Access: Mixed

	31	30	29	28	27	26	25	24	23	22	21	20	19			16
R	PTS		—	PTW	PSPD		—	PFSC	PHCD	WKOC	WKDS	WLCN	PTC			
W																
Reset (MPH)	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
Reset (DR)	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PIC		PO	PP	LS		—	PR	SUSP	FPR	OCC	OCA	PEC	PE	CSC	CCS
W											w1c		w1c		w1c	
Reset (MPH)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Reset (DR)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-22. Port Status and Control (PORTSC*n*)

16.3.2.21, 16-56 Change the offset of Figure 16-33 to Base +1B4h.

16.3.2.28, 16-64 — Change the base offset to ‘Base +410h’ and remove the names on bits 28-30 because they should be reserved. Replace Figure 16-34 and Table 16-41 with the following:

Base +410h

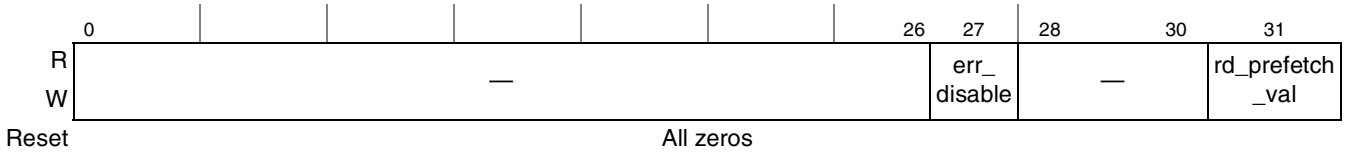


Figure 16-34. System Interface Control Register (SI_CTRL)

Table 16-41. SI_CTRL Register Field Descriptions

Bits	Name	Description
0–26	—	Reserved, should be cleared
27	err_disable	When this bit is HIGH, it causes the controller to ignore system bus errors. If LOW the controller responds according to the values set in the SEI bit of the USBSTS register and the SEE bit of the USBINT register. 1 Disable. 0 Enable.
28–30	—	Reserved, should be cleared
31	rd_prefetch_val	This register bit 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 set. Otherwise, this value should be left cleared. 1 64-byte fetch. 0 32-byte fetch.

16.3.2.29, 16-65 — In Figure 16-35, change the base offset to: Base +40Ch.

16.6.1, 16-90 — Remove the fifth bullet, ‘Program the CTRLDSSEGMENT register with 4-Gigabyte segment where all of the interface data structures are allocated.’ According to EHCI standard, CTRLDSSEGMENT is used to allow 64-bit addressing on system bus. The ARC/TDI controller implementation does not allow this, so this register is not implemented.

17.4.5.3, 17-19 — In Figure 17-10, the address bits should read ADDR[22:29], and not ADDR[12:29].

19.4, 19-9 — The access of the register SPIE event register (SPIE) should be changed from ‘R/W’ to ‘Mixed.’

19.4.1.2, 19-13 — The SPI Event Register (SPIE) bits NE and NF are not affected by a write. The third sentence should be replaced with:

Most SPIE bits can be cleared by writing a ‘1’.

Also, the NE and NF bits in Figure 19-7 should be changed to read only and the access changed to ‘Mixed:’

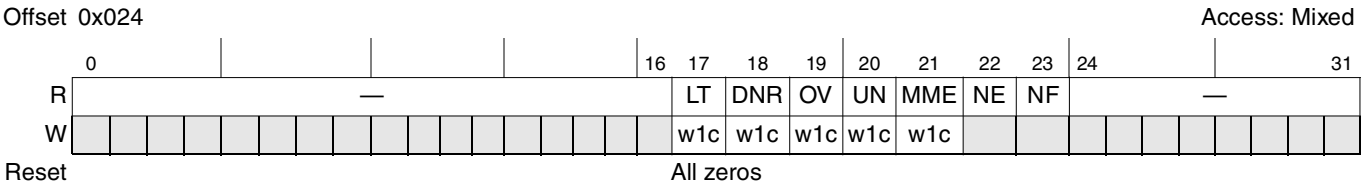


Figure 19-7. SPIE—SPI Event Register

22.4, 22-3 In the last row of Table 22-2, the reserved register space should begin at offset 0x14.

22.4.3, 22-5 Replace Figure 22-5 with the following:

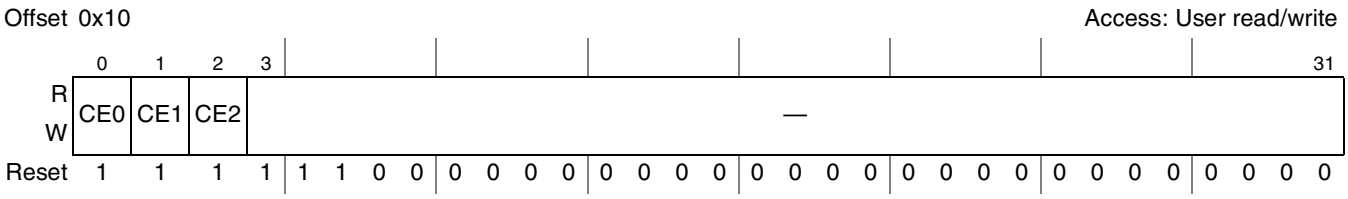


Figure 22-5. DLL Clock Register

22.4.3, 22-5 Replace Table 22-5 with the following:

Table 22-5. DLLCK Field Description

Bits	Name	Description
0	CE0	Enable/Disable LCLK[0] signal clock out 0 Disable LCLK[0] 1 Enable LCLK[0]
1	CE1	Enable/Disable LCLK[1] signal clock out 0 Disable LCLK[1] 1 Enable LCLK[1]
2	CE2	Enable/Disable LCLK[2] signal clock out 0 Disable LCLK[2] 1 Enable LCLK[2]
3–31	—	Reserved

Glossary of Terms and Abbreviations

The glossary contains an alphabetical list of terms, phrases, and abbreviations used in this reference manual.

A

Architecture. A detailed specification of requirements for a processor or computer system. It does not specify details of how the processor or computer system must be implemented; instead it provides a template for a family of compatible *implementations*.

Atomic access. A bus access that attempts to be part of a read-write operation to the same address uninterrupted by any other access to that address (the term refers to the fact that the transactions are indivisible). The PowerPC architecture implements atomic accesses through the **lwarx**/**stwcx.** instruction pair.

Autobaud. The process of determining a serial data rate by timing the width of a single bit.

B

Beat. A single state on the e300 bus interface that may extend across multiple bus cycles. A e300 transaction can be composed of multiple address or data *beats*.

Big-endian. A byte-ordering method in memory where the address *n* of a word corresponds to the *most-significant byte*. In an addressed memory word, the bytes are ordered (left to right) 0, 1, 2, 3, with 0 being the *most-significant byte*. See *Little-endian*.

Boundedly undefined. A characteristic of certain operation results that are not rigidly prescribed by the PowerPC architecture. Boundedly- undefined results for a given operation may vary among implementations and between execution attempts in the same implementation.

Although the architecture does not prescribe the exact behavior for when results are allowed to be boundedly undefined, the results of executing instructions in contexts where results are allowed to be boundedly undefined are constrained to ones that could have been achieved by executing an arbitrary sequence of defined instructions, in valid form, starting in the state the machine was in before attempting to execute the given instruction.

Breakpoint. A programmable event that forces the core to take a breakpoint exception.

Burst. A multiple-beat data transfer whose total size is typically equal to a cache block.

Bus clock. Clock that causes the bus state transitions.

Bus master. The owner of the address or data bus; the device that initiates or requests the transaction.

C

Cache. High-speed memory containing recently accessed data or instructions (subset of main memory).

Cache block. A small region of contiguous memory that is copied from memory into a *cache*. The size of a cache block may vary among processors; the maximum block size is one *page*. In PowerPC processors, *cache coherency* is maintained on a cache-block basis. Note that the term ‘cache block’ is often used interchangeably with ‘cache line.’

Cache coherency. An attribute wherein an accurate and common view of memory is provided to all devices that share the same memory system. Caches are coherent if a processor performing a read from its cache is supplied with data corresponding to the most recent value written to memory or to another processor’s cache.

Cache flush. An operation that removes from a cache any data from a specified address range. This operation ensures that any modified data within the specified address range is written back to main memory. This operation is generated typically by a Data Cache Block Flush (**dcbf**) instruction.

Caching-inhibited. A memory update policy in which the *cache* is bypassed and the load or store is performed to or from main memory.

Cast out. A *cache block* that must be written to memory when a cache miss causes a cache block to be replaced.

Changed bit. One of two *page history bits* found in each *page table entry* (PTE). The processor sets the changed bit if any store is performed into the *page*. See also *Page access history bits* and *Referenced bit*.

Clean. An operation that causes a cache block to be written to memory, if modified, and then left in a valid, unmodified state in the cache.

Clear. To cause a bit or bit field to register a value of zero. See also *Set*.

Context synchronization. An operation that ensures that all instructions in execution complete past the point where they can produce an *exception*, that all instructions in execution complete in the context in which they began execution, and that all subsequent instructions are *fetched* and executed in the new context. Context synchronization may result from executing specific instructions (such as **isync** or **rfi**) or when certain events occur (such as an exception).

Copy-back operation. A cache operation in which a cache line is copied back to memory to enforce cache coherency. Copy-back operations consist of snoop push-out operations and cache cast-out operations.

D

Direct-mapped cache. A cache in which each main memory address can appear in only one location within the cache, operates more quickly when the memory request is a cache hit.

Double data rate. Memory that allows data transfers at the start and end of a clock cycle. thereby doubling the data rate.

E

Effective address (EA). The 32-bit address specified for a load, store, or an instruction fetch. This address is then submitted to the MMU for translation to either a *physical memory* address or an I/O address.

Exclusive state. MEI state (E) in which only one caching device contains data that is also in system memory.

F

Frame-check sequence (FCS). Specifies the standard 32-bit cyclic redundancy check (CRC) obtained using the standard CCITT-CRC polynomial on all fields except the preamble, SFD, and CRC.

Fetch. Retrieving instructions from either the cache or main memory and placing them into the instruction queue.

Flush. An operation that causes a cache block to be invalidated and the data, if modified, to be written to memory.

G

General-purpose register (GPR). Any of the 32 registers in the general-purpose register file. These registers provide the source operands and destination results for all integer data manipulation instructions. Integer load instructions move data from memory to GPRs and store instructions move data from GPRs to memory.

Gigabit media-independent interface (GMII) sublayer. Sublayer that provides a standard interface between the MAC layer and the physical layer for 1000-Mbps operation. It isolates the MAC layer and the physical layer, enabling the MAC layer to be used with various implementations of the physical layer.

Guarded. The guarded attribute pertains to out-of-order execution. When a page is designated as guarded, instructions and data cannot be accessed out-of-order.

H

Harvard architecture. An architectural model featuring separate caches and other memory management resources for instructions and data.

I

IEEE 754. A standard written by the Institute of Electrical and Electronics Engineers that defines operations and representations of binary floating-point numbers.

Illegal instructions. A class of instructions that are not implemented for a particular PowerPC processor. These include instructions not defined by the PowerPC architecture. In addition, for 32-bit implementations, instructions that are defined only for 64-bit implementations are considered to be illegal instructions. For 64-bit implementations instructions that are defined only for 32-bit implementations are considered to be illegal instructions.

Implementation. A particular processor that conforms to the PowerPC architecture, but may differ from other architecture-compliant implementations for example in design, feature set, and implementation of *optional* features. The PowerPC architecture has many different implementations.

Inbound windows. Mappings that perform address translation from the external address space to the local address space, attach attributes and transaction types to the transaction, and map the transaction to its target interface.

Inter-packet gap. The gap between the end of one Ethernet packet and the beginning of the next transmitted packet.

Integer unit. An execution unit in the core responsible for executing integer instructions.

In-order. An aspect of an operation that adheres to a sequential model. An operation is said to be performed in-order if, at the time that it is performed, it is known to be required by the sequential execution model. See *Out-of-order.*

Instruction latency. The total number of clock cycles necessary to execute an instruction and make ready the results of that instruction.

K

Kill. An operation that causes a *cache block* to be invalidated without writing any modified data to memory.

L

Latency. The number of clock cycles necessary to execute an instruction and make ready the results of that execution for a subsequent instruction.

L2 cache. Level-2 cache. See *Secondary cache*.

Least-significant bit (lsb). The bit of least value in an address, register, field, data element, or instruction encoding.

Least-significant byte (LSB). The byte of least value in an address, register, data element, or instruction encoding.

Little-endian. A byte-ordering method in memory where the address *n* of a word corresponds to the *least-significant byte*. In an addressed memory word, the bytes are ordered (left to right) 3, 2, 1, 0, with 3 being the *most-significant byte*. See *Big-endian*.

Local access window. Mapping used to translate a region of memory to a particular target interface, such as the DDR SDRAM controller or the PCI controller. The local memory map is defined by a set of eight local access windows. The size of each window can be configured from 4 Kbytes to 2 Gbytes.

M

Media access control (MAC) sublayer. Sublayer that provides a logical connection between the MAC and its peer station. Its primary responsibility is to initialize, control, and manage the connection with the peer station.

Medium-dependent interface (MDI) sublayer—Sublayer that defines different connector types for different physical media and PMD devices.

Media-independent interface (MII) sublayer. Sublayer that provides a standard interface between the MAC layer and the physical layer for 10/100-Mbps operations. It isolates the MAC layer and the physical layer, enabling the MAC layer to be used with various implementations of the physical layer.

MEI (modified/exclusive/invalid). *Cache coherency* protocol used to manage caches on different devices that share a memory system. Note that the PowerPC architecture does not specify the implementation of a MEI protocol to ensure cache coherency.

Memory access ordering. The specific order in which the processor performs load and store memory accesses and the order in which those accesses complete.

Memory-mapped accesses. Accesses whose addresses use the page or block address translation mechanisms provided by the MMU and that occur externally with the bus protocol defined for memory.

Memory coherency. An aspect of caching in which it is ensured that an accurate view of memory is provided to all devices that share system memory.

Memory consistency. Refers to agreement of levels of memory with respect to a single processor and system memory (for example, on-chip cache, secondary cache, and system memory).

Memory management unit (MMU). The functional unit that is capable of translating an *effective* (logical) *address* to a physical address, providing protection mechanisms, and defining caching methods.

Modified state. MEI state (M) in which one, and only one, caching device has the valid data for that address. The data at this address in external memory is not valid.

Most-significant bit (msb). The highest-order bit in an address, registers, data element, or instruction encoding.

Most-significant byte (MSB). The highest-order byte in an address, registers, data element, or instruction encoding.

N

NaN. An abbreviation for not a number; a symbolic entity encoded in floating-point format. There are two types of NaNs—signaling NaNs and quiet NaNs.

No-op. No-operation. A single-cycle operation that does not affect registers or generate bus activity.

O

Outbound Windows. Mappings that perform address translations from local 32-bit address space to the address spaces of PCI1 or PCI2, which may be much larger than the local space. Outbound windows also map attributes such as transaction type or priority level.

P

Packet. A unit of binary data that can be routed through a network. Sometimes packet is used to refer to the frame plus the preamble and start frame delimiter (SFD).

Page. A region in memory. The OEA defines a page as a 4-Kbyte area of memory, aligned on a 4-Kbyte boundary.

Page access history bits. The *changed* and *referenced* bits in the PTE keep track of the access history within the page. The referenced bit is set by the MMU whenever the page is accessed for a read or write operation. The changed bit is set when the page is stored into. See *Changed bit* and *Referenced bit.*

Page fault. A page fault is a condition that occurs when the processor attempts to access a memory location that does not reside within a *page* not currently resident in *physical memory.* On PowerPC processors, a page fault exception condition occurs when a matching, valid *page table entry* (PTE[V] = 1) cannot be located.

Page table. A table in memory is comprised of *page table entries*, or PTEs. It is further organized into eight PTEs per PTEG (page table entry group). The number of PTEGs in the page table depends on the size of the page table (as specified in the SDR1 register).

Page table entry (PTE). Data structures containing information used to translate *effective address* to physical address on a 4-Kbyte page basis. A PTE consists of 8 bytes of information in a 32-bit processor and 16 bytes of information in a 64-bit processor.

Physical coding sublayer (PCS). Sublayer responsible for encoding and decoding data stream to and from the MAC sublayer. Medium (1000BASEX) 8B/10B coding is used for fiber. Medium (1000BASET) 8B1Q coding is used for unshielded twisted pair (UTP).

Physical medium attachment (PMA) sublayer. Sublayer responsible for serializing code groups into a bit stream suitable for serial bit-oriented physical devices (SERDES) and vice versa. Synchronization is also performed for proper data decoding in this sublayer. The PMA sits between the PCS and the PMD sublayers. For fiber medium (1000BASEX) the interface on the PMD side of the PMA is a one-bit 1250 MHz signal, while on the PMA's PCS side the interface is a ten-bit interface (TBI) at 125 MHz. The TBI is an alternative to the GMII interface. If the TBI is used the gigabit Ethernet controller must be capable of performing the PCS function. For UTP medium, the PMD interface side of the PMA consists of four pair of 62.5-MHz PAM5 encoded signals, while the PCS side provides the 1250-Mbps input to a 8B1Q4 PCS.

Physical medium dependent (PMD) sublayer. Sublayer responsible for signal transmission. The typical PMD functionality includes amplifier, modulation, and wave shaping. Different PMD devices may support different media.

Physical memory. The actual memory that can be accessed through the system's memory bus.

Pipelining. A technique that breaks operations, such as instruction processing or bus transactions, into smaller distinct stages or tenures (respectively) so that a subsequent operation can begin before the previous one has completed.

Precise exceptions. A category of exception for which the pipeline can be stopped so instructions that preceded the faulting instruction can complete and subsequent instructions can be flushed and redispatched after exception handling has completed. See *Imprecise exceptions.*

Primary opcode. The most-significant 6 bits (bits 0–5) of the instruction encoding that identifies the type of instruction.

Program order. The order of instructions in an executing program. More specifically, this term is used to refer to the original order in which program instructions are fetched into the instruction queue from the cache.

Protection boundary. A boundary between *protection domains*.

Protection domain. A protection domain is a segment, a virtual page, a BAT area, or a range of unmapped effective addresses. It is defined only when the appropriate relocate bit in the MSR (IR or DR) is 1.

Q

Quad word. A group of 16 contiguous locations starting at an address divisible by 16.

Quiesce. To come to rest. The processor is said to quiesce when an exception is taken or a **sync** instruction is executed. The instruction stream is stopped at the decode stage and executing instructions are allowed to complete to create a controlled context for instructions that may be affected by out-of-order, parallel execution. See *Context synchronization*.

R

rA. The **r**A instruction field is used to specify a GPR to be used as a source or destination.

rB. The **r**B instruction field is used to specify a GPR to be used as a source.

rD. The **r**D instruction field is used to specify a GPR to be used as a destination.

rS. The **r**S instruction field is used to specify a GPR to be used as a source.

Record bit. Bit 31 (or the Rc bit) in the instruction encoding. When it is set, updates the condition register (CR) to reflect the result of the operation.

Reconciliation sublayer. Sublayer that maps the terminology and commands used in the MAC layer into electrical formats appropriate for the physical layer entities.

Referenced bit. One of two *page history bits* found in each *page table entry.* The processor sets the *referenced bit* whenever the page is accessed for a read or write. See also *Page access history bits.*

Reservation. The processor establishes a reservation on a *cache block* of memory space when it executes an **lwarx** instruction to read a memory semaphore into a GPR.

Reservation station. A buffer between the dispatch and execute stages that allows instructions to be dispatched even though the results of instructions on which the dispatched instruction may depend are not available.

RISC (reduced instruction set computing). An *architecture* characterized by fixed-length instructions with nonoverlapping functionality and by a separate set of load and store instructions that perform memory accesses.

S

Secondary cache. A cache memory that is typically larger and has a longer access time than the primary cache. A secondary cache may be shared by multiple devices. Also referred to as L2, or level-2, cache.

Set (*v*)**.** To write a nonzero value to a bit or bit field; the opposite of *clear.* The term ‘set’ may also be used to generally describe the updating of a bit or bit field.

Set (*n*)**.** A subdivision of a *cache*. Cacheable data can be stored in a given location in one of the sets, typically corresponding to its lower-order address bits. Because several memory locations can map to the same location, cached data is typically placed in the set whose *cache block* corresponding to that address was used least recently. See *Set-associative*.

Set-associative. Aspect of cache organization in which the cache space is divided into sections, called *sets*. The cache controller associates a particular main memory address with the contents of a particular set, or region, within the cache.

Slave. The device addressed by a master device. The slave is identified in the address tenure and is responsible for supplying or latching the requested data for the master during the data tenure.

Snooping. Monitoring addresses driven by a bus master to detect the need for coherency actions.

Snoop push. Response to a snooped transaction that hits a modified cache block. The cache block is written to memory and made available to the snooping device.

Stall. An occurrence when an instruction cannot proceed to the next stage.

Sticky bit. A bit that when *set* must be cleared explicitly.

Superscalar machine. A machine that can issue multiple instructions concurrently from a conventional linear instruction stream.

Supervisor mode. The privileged operation state of a processor. In supervisor mode, software, typically the operating system, can access all control registers and can access the supervisor memory space, among other privileged operations.

Synchronization. A process to ensure that operations occur strictly *in order*. See *Context synchronization* and *Execution synchronization.*

Synchronous exception. An *exception* that is generated by the execution of a particular instruction or instruction sequence. There are two types of synchronous exceptions, *precise* and *imprecise*.

System memory. The physical memory available to a processor.

T

Time-division multiplex (TDM). A single serial channel used by several channels taking turns.

Tenure. The period of bus mastership. For the e300, there can be separate address bus tenures and data bus tenures. A tenure consists of three phases: arbitration, transfer, and termination.

TLB (translation lookaside buffer). A cache that holds recently-used *page table entries*.

Throughput. The measure of the number of instructions that are processed per clock cycle.

Transaction. A complete exchange between two bus devices. A transaction is typically comprised of an address tenure and one or more data tenures, which may overlap or occur separately from the address tenure. A transaction may be minimally comprised of an address tenure only.

Transfer termination. Signal that refers to both signals that acknowledge the transfer of individual beats (of both single-beat transfer and individual beats of a burst transfer) and to signals that mark the end of the tenure.

U

User mode. The operating state of a processor used typically by application software. In user mode, software can access only certain control registers and can access only

user memory space. No privileged operations can be performed. Also referred to as problem state.

V

Virtual address. An intermediate address used in the translation of an *effective address* to a physical address.

Virtual memory. The address space created using the memory management facilities of the processor. Program access to *virtual memory* is possible only when it coincides with *physical memory.*

W

Way. A location in the cache that holds a cache block, its tags, and status bits.

Word. A 32-bit data element.

Write-back. A cache memory update policy in which processor write cycles are directly written only to the cache. External memory is updated only indirectly, for example, when a modified cache block is *cast out* to make room for newer data.

Write-through. A cache memory update policy in which all processor write cycles are written to both the cache and memory.

Index 1
Register Index (Memory-mapped registers)

A

ACR (arbiter configuration register), 2
AEADR (arbiter event address register), 9
AEATR (arbiter event attributes register), 7
AER (arbiter event register), 5
AERR (arbiter event response register), 10
AESUDSR (AESU data size register), 71
AESUEMR (AESU end of message register), 76
AESUICR (AESU interrupt control register), 75
AESUISR (AESU interrupt status register), 74
AESUKSR (AESU key size register), 71
AESUMR (AESU mode register), 68
AESURCR (AESU reset control register), 72
AESUSR (AESU status register), 73
AFEUDSR (AFEU context/data size register), 45
AFEUEMR (AFEU end of message register), 50
AFEUICR (AFEU interrupt control register), 49
AFEUISR (AFEU interrupt status register), 47
AFEUK0 (AFEU key register 0), 51
AFEUK1 (AFEU key register 1), 51
AFEUKSR (AFEU key size register), 44
AFEUMR (AFEU mode register), 43
AFEURCR (AREU reset control register), 46
AFEUSR (AFEU status register), 46
AGE_CNT_THRESH (USB age count threshold register), 54
AIDR (arbiter interrupt definition register), 6
ALTCBAR (alternate configuration base address register), 7
AMR (arbiter mask register), 7
ANA (TSEC AN advertisement register), 93
ANEX (TSEC AN expansion register), 96
ANLPANP (TSEC AN link partner ability next page register), 98
ANLPBPA (TSEC AN link partner base page ability register), 95
ANNPT (TSEC AN next page transmit register), 97
ASYNCLISTADDR (USB current asynchronous list address register), 32
ASYNCTTSTS (USB transaction translator asynchronous buffer status), 34
ATR (arbiter timers register), 4
ATTR (TSEC attribute register), 89

B

BR*n* (LBC base registers 0–7), 11
BURSTSIZE (USB master interface data burst size register), 34

C

CAM1 (TSEC carry mask register 1), 85
CAM2 (TSEC carry mask register 2), 86
CAPLENGTH (USB capability registers length), 19
CAPTURE_ADDRESS (DDR memory error address capture register), 37
CAPTURE_ATTRIBUTES (DDR memory error attributes capture register), 36
CAPTURE_DATA_HI (DDR memory data path read capture high register), 32
CAPTURE_DATA_LO (DDR memory data path read capture low register), 33
CAPTURE_ECC (DDR memory data path read capture ECC register), 33
CAR1 (TSEC carry register 1), 82
CAR2 (TSEC carry register 2), 83
CCCR (crypto-channel configuration register), 82
CCPSR (crypto-channel pointer status register), 85
CDPR (crypto-channel current descriptor pointer register), 90
CONFIGFLAG (USB configure flag register), 39
CONTROL (USB general purpose control register), 57
CR (TSEC control register), 91
CRBPTR (TSEC current receive buffer descriptor pointer register), 43
CS*n*BNDS (DDR chip select 0–3 bounds registers), 10
CS*n*CONFIG (DDR chip select 0–3 configuration registers), 11
CTBPTR (TSEC current transmit buffer descriptor pointer register), 36
CTRLDSSEGMENT (USB control data structure segment register), 31

D

DATA_ERR_INJECT_HI (DDR memory data path error injection mask high register), 31
DATA_ERR_INJECT_LO (DDR memory data path error injection mask low register), 31
DCCPARAMS (USB device controller capability parameters), 22
DCIVERSION (USB device controller interface version), 22

DDR_DATA_INIT (DDR SDRAM data initialization register), 28
DDR_INIT_ADDR (DDR initialization address register), 29
DDR_IP_REV1 (DDR IP block revision register 1), 30
DDR_IP_REV2 (DDR IP block revision register 2), 30
DDR_SDRAM_CFG (DDR SDRAM control configuration register), 20
DDR_SDRAM_CFG_2 (DDR SDRAM control configuration register 2), 22
DDR_SDRAM_CLK_CNTL (DDR SDRAM clock control register), 28
DDR_SDRAM_INTERVAL (DDR SDRAM interval configuration register), 27
DDR_SDRAM_MD_CNTL (DDR SDRAM mode control register), 25
DDR_SDRAM_MODE (DDR SDRAM mode configuration register), 24
DDR_SDRAM_MODE_2 (DDR SDRAM mode 2 configuration register), 24
DDRCDR (DDR control driver register), 28
DDRDSR (DDR debus status register), 30
DDRLAWAR0–DDRLAWAR1 (DDR local access window *n* attributes registers), 13
DDRLAWBAR0–DDRLAWBAR1 (DDR local access window *n* base address registers), 12
DEUDSR (DEU data size register), 36
DEUEUG (DEU EU-Go register), 41
DEUICR (DEU interrupt control register), 40
DEUISR (DEU interrupt status register), 38
DEUIV (DEU IV register), 42
DEUK1—DEUK3 (DEU key registers 1—3), 42
DEUK2 (DEU key register 2), 42
DEUK3 (DEU key register 3), 42
DEUKSR (DEU key size register), 36
DEUMR (DEU mode register), 35
DEURCR (DEU reset control register), 37
DEUSR (DEU status register), 37
DEVICEADDR (USB device address register), 32
DID (device ID configuration register), 31
DLLCK (DLL clock register), 5
DLLOVR (DLL override register), 4
DLLSR (DLL status register), 4
DMABCR*n* (DMA byte count register), 17
DMACDAR*n* (DMA current descriptor address register), 15
DMACTRL (TSEC DMA control register), 28
DMADAR*n* (DMA destination address register), 16
DMAGSR*n* (DMA general status register), 18
DMAMR*n* (DMA mode register), 12
DMANDAR*n* (DMA next descriptor address register), 17
DMASAR*n* (DMA source address register), 16
DMASR*n* (DMA status register), 14
DTCR (discard timer control register), 6

E

ECC_ERR_INJECT (DDR memory data path error injection mask ECC register), 32
ECNTRL (TSEC Ethernet control register), 25
EDIS (TSEC error disabled register), 24
ENDPOINTLISTADDR (USB endpoint list address register), 33
ENDPTCOMPLETE (USB endpoint complete register), 50
ENDPTCTRL0 (USB endpoint control register 0), 51
ENDPTCTRL*n* (USB endpoint control register *n*), 52
ENDPTFLUSH (USB endpoint flush register), 49
ENDPTPRIME (USB endpoint initialization register), 48
ENDPTSETUPSTAT (USB endpoint setup status register), 48
ENDPTSTATUS (USB endpoint status register), 50
ERR_DETECT (DDR memory error detect register), 33
ERR_DISABLE (DDR memory error disable register), 34
ERR_INJECT, *see* ECC_ERR_INJECT
ERR_INT_EN (DDR memory error interrupt enable register), 35
ERR_SBE (DDR single-bit ECC memory error management register), 37
EUASR (EU assignment status register), 93
EXST (TSEC extended status register), 98

F

FIFO_TX_SP (TSEC FIFO transmit space available register), 31
FIFO_TX_STARVE (TSEC FIFO transmit starve register), 31
FIFO_TX_STARVE_SHUTOFF (TSEC FIFO transmit starve shutoff register), 32
FIFO_TX_THR (TSEC FIFO transmit threshold register), 30
FRINDEX (USB frame index register), 30

G

GADDR*n* (TSEC group address registers 0–7), 88
GP*n*DAT (GPIO*n* data register), 4
GP*n*DIR (GPIO*n* direction register), 3
GP*n*ICR (GPIO*n* interrupt control register), 6
GP*n*IER (GPIO*n* interrupt event register), 5
GP*n*IMR (GPIO*n* interrupt mask register), 5
GP*n*ODR (GPIO*n* open drain register), 4
GTCFR*n* (global timers configuration registers), 57
GTCNR1–GTCNR4 (global timers counter registers), 63
GTCPR1–GTCPR4 (global timers capture registers), 62
GTEVR1–GTEVR4 (global timers event registers), 63
GTMDR1–GTMDR4 (global timers mode registers), 60
GTPSR1–GTPSR4 (global timers prescale registers), 64
GTRFR1–GTRFR4 (global timers reference registers), 62

H

HAFDUP (TSEC half-duplex register), 52
HCCPARAMS (USB host controller capability parameters), 21
HCIVERSION (USB host controller interface version), 19
HCSPARAMS (USB host controller structural parameters), 20

I

I2C*n*ADR (I^2C address register), 5
I2C*n*CR (I^2C control register), 6
I2C*n*DFSRR (I^2C digital filter sampling rate register), 10
I2C*n*DR (I^2C data register), 9
I2C*n*FDR (I^2C frequency divider register), 5
I2C*n*SR (I^2C status register), 8
IADDR*n* (TSEC individual address registers 0–7), 87
ICR (interrupt clear register), 96
ID (identification register), 98
IDR (inbound doorbell register), 9
IEVENT (TSEC interrupt event register), 19
IFSTAT (TSEC interface status register), 58
IMASK (TSEC interrupt mask register), 22
IMIMR (inbound message interrupt mask register), 11
IMISR (inbound message interrupt status register), 9
IMMRBAR (internal memory map registers base address register), 5
IMR (interrupt mask register), 94
IMR*n* (inbound message registers), 7
IPGIFG (TSEC inter-packet gap/inter-frame gap register), 51
ISR (interrupt status register), 96

J

JD (TSEC jitter diagnostics register), 99

L

LBCR (LBC configuration register), 29
LBLAWAR0–LBLAWAR3 (LBIU local access window *n* attributes registers), 8
LBLAWBAR0–LBLAWBAR1 (LBIU local access window *n* base address registers), 7
LCRR (LBC clock ratio register), 30
LSDMR (LBC SDRAM machine mode register), 22
LSRT (LBC SDRAM refresh timer), 25
LTEAR (LBC transfer error address register), 29
LTEATR (LBC transfer error attributes register), 28
LTEDR (LBC transfer error check disable register), 27
LTEIR (LBC transfer error interrupt enable register), 27
LTESR (LBC transfer error status register), 26
LURT (UPM refresh timer), 24

M

MACCFG1 (TSEC MAC configuration register 1), 48
MACCFG2 (TSEC MAC configuration register 2), 49
MACSTNADDR1 (TSEC station address part 1 register), 58
MACSTNADDR2 (TSEC station address part 2 register), 59
MAR (UPM address register), 18
MAXFRM (TSEC maximum frame length register), 53
MCKENR (MCK clock enable register), 41
MCR (master control register), 98
MDEUDSR (MDEU data size register), 56
MDEUEUG (MDEU EU-Go register), 61
MDEUICR (MDEU interrupt control register), 59
MDEUICVSR (MDEU ICV size register), 60
MDEUISR (MDEU interrupt status register), 58
MDEUKSR (MDEU key size register), 55
MDEUMR (MDEU mode register), 51
MDEURCR (MEDU reset control register), 56
MDEUSR (MDEU status register), 57
MDR (UPM data register), 22
MIIMADD (TSEC MII management address register), 55
MIIMCFG (TSEC MII management configuration register), 53
MIIMCOM (TSEC MII management command register), 54
MIIMCON (TSEC MII management control register), 56
MIIMIND (TSEC MII management indicator register), 57
MIIMSTAT (TSEC MII management status register), 56
MINFLR (TSEC minimum frame length register), 26
M*n*MR (LBC UPM machine A–C mode registers), 19–21
MRBLR (TSEC maximum receive buffer length register), 43
MRTPR (LBC memory refresh timer prescaler register), 21

O

OCCR (output clock control register), 38
ODR*n* (outbound doorbell register), 8
OMIMR (outbound message interrupt mask register), 6
OMISR (outbound message interrupt register), 4
OMR*n* (outbound message registers), 7
OR*n* (LBC options registers 0–7), 12–18
OSTBD (TSEC out-of-sequence TxBD register), 37
OSTBDP (TSEC out-of-sequence Tx data buffer pointer register), 39
OTGSC (USB On-the-Go status and control register), 44

P

PCI configuration registers
see General Index, PCI, registers
PCI_CONFIG_ADDRESS (PCI configuration address register), 16
PCI_CONFIG_DATA (PCI configuration data register), 18
PCI_EACR (PCI error address capture register), 23
PCI_EATCR (PCI error attributes capture register), 21

PCI_ECDR (PCI error capture disable register), 19
PCI_ECR (PCI error control register), 25
PCI_EDHCR (PCI error data high capture register), 24
PCI_EDLCR (PCI error data low capture register), 24
PCI_EEACR (PCI error extended address capture register), 23
PCI_EER (PCI error enable register), 20
PCI_ESR (PCI error status register), 18
PCI_GCR (PCI general control register), 24
PCI_GSR (PCI general status register), 26
PCI_INT_ACK (PCI interrupt acknowledge register), 18
PCILAWAR0–PCILAWAR1 (PCI local access window *n* attributes registers), 10
PCILAWBAR0–PCILAWBAR1 (PCI local access window *n* base address registers), 9
PERIODICLISTBASE (USB periodic frame list base address register), 31
PIBAR*n* (PCI inbound base address registers), 27
PIEBAR*n* (PCI inbound extended base address registers), 27
PITAR*n* (PCI inbound translation address registers), 26
PIWAR*n* (PCI inbound window attribute registers), 28
PKEUABS (PKEU AB size register), 28
PKEUDSR (PKEU data size register), 29
PKEUEUG (PKEU EU-Go register), 33
PKEUICR (PKEU interrupt control register), 32
PKEUISR (PKEU interrupt status register), 31
PKEUKSR (PKEU key size register), 28
PKEUMR (PKEU mode register), 26
PKEURCR (PKEU reset control register), 29
PKEUSR (PKEU status register), 30
PMCCR (power management controller configuration register), 69
PMCER (power management controller event register), 70
PMCMR (power management controller mask register), 71
PMCR (power management control register), 5
POBAR*n* (PCI outbound base address registers), 3
POCMR*n* (PCI outbound comparison mask registers), 4
PORTSC*n* (USB port status and control registers), 39
POTAR*n* (PCI outbound translation address registers), 3
PRI_CTRL (USB priority control register), 56
PTCNR (periodic interval timer control register), 47
PTCTR (periodic interval timer counter register), 49
PTEVR (periodic interval timer event register), 50
PTLDR (periodic interval timer load register), 48
PTPSR (periodic interval timer prescale register), 49
PTV (TSEC pause time value register), 27

R

RALN (TSEC receive alignment error counter), 68
RBASE (TSEC receive descriptor base address register), 44
RBCA (TSEC receive broadcast packet counter register), 66
RBDLEN (TSEC RxBD data length register), 41
RBPTR (TSEC receive buffer descriptor pointer register), 44
RBYT (TSEC receive byte counter), 64
RCDE (TSEC receive code error counter register), 69
RCER (reset control enable register), 36
RCR (reset control register), 36
RCSE (TSEC receive carrier sense error counter register), 69
RCTRL (TSEC receive control register), 40
RCWHR (reset configuration word high register), 14, 32
RCWLR (reset configuration word low register), 11, 32
RDRP (TSEC receive dropped packet counter register), 72
RFCS (TSEC receive FCS error counter register), 65
RFLR (TSEC receive frame length error counter register), 68
RFRG (TSEC receive fragments counter register), 71
RJBR (TSEC receive jabber counter register), 71
RMCA (TSEC receive multicast packet counter register), 65
RMR (reset mode register), 34
RNGDSR (RNG data size register), 64
RNGEUG (RNG EU-Go register), 68
RNGICR (RNG interrupt control register), 67
RNGISR (RNG interrupt status register), 66
RNGMR (RNG mode register), 63
RNGRCR (RNG reset control register), 65
RNGSR (RNG status register), 65
ROVR (TSEC receive oversize packet counter), 70
RPKT (TSEC receive packet counter), 64
RPR (reset protection register), 35
RSR (reset status register), 33
RSTAT (TSEC receive status register), 41
RTALR (real time counter alarm register), 43
RTCNR (real time counter control register), 40
RTCTR (real time counter register), 42
RTEVR (real time counter event register), 42
RTLDR (real time counter load register), 41
RTPSR (real time counter prescale register), 41
RUND (TSEC receive undersize packet counter register), 70
RXCF (TSEC receive control frame packet counter register), 66
RXIC (TSEC receive interrupt coalescing configuration register), 42
RXPF (TSEC receive pause frame packet counter register), 67
RXUO (TSEC receive unknown opcode packet counter register), 67

S

SCCR (system clock control register), 40
SCVCR (system critical interrupt vector register), 27
SECNR (system external interrupt control register), 21
SEFCR (system external interrupt force register), 26
SEMSR (system external interrupt mask register), 20
SEPNR (system external interrupt pending register), 18
SERCR (system error control register), 24

SERFR (system error force register), 26
SERMR (system error mask register), 23
SERSR (system error status register), 22
SGPRH (system general purpose register high), 17
SGPRL (system general purpose register low), 17
SI_CTRL (USB system interface control register), 56
SICFR (system global interrupt configuration register), 8
SICNR (system internal interrupt control register), 16
SICRH (system I/O configuration register high), 24
SICRL (system I/O configuration register low), 21
SIFCR_H (system internal interrupt force register high), 25
SIFCR_L (system internal interrupt force register low), 25
SIMSR_H (system internal interrupt mask register high), 15
SIMSR_L (system internal interrupt mask register low), 15
SIPNR_H (system internal interrupt pending register high), 11
SIPNR_L (system internal interrupt pending register low), 11
SIPRR_A (system internal interrupt group A priority register), 14
SIPRR_D (system internal interrupt group D priority register), 14
SIVCR (system global interrupt vector register), 9
SMPRR_A (system mixed interrupt group A priority register), 18
SMPRR_B (system mixed interrupt group B priority register), 19
SMVCR (system management interrupt vector register), 27
SNOOP*n*, 53
SPCOM (SPI command register), 14
SPCR (system priority and configuration register), 19
SPIE (SPI event register), 11
SPIM (SPI mask register), 13
SPIRD (SPI receive data hold register), 15
SPITD (SPI transmit data hold register), 14
SPMODE (SPI mode register), 9
SPRIDR (system part and revision ID register), 18
SPRM (system PLL mode register), 37
SR (TSEC status register), 92
SWCNR (system watchdog count register), 34
SWCRR (system watchdog control register), 33
SWSRR (system watchdog service register), 34

T

TBASE (TSEC transmit descriptor base address register), 37
TBCA (TSEC transmit broadcast packet counter register), 74
TBDLEN (TSEC TxBD data length register), 34
TBICON (TSEC TBI control register), 100
TBIPA (TSEC TBI physical address register), 29
TBPTR (TSEC transmit buffer descriptor pointer register), 36
TBYT (TSEC transmit byte counter register), 72
TCTRL (TSEC transmit control register), 33
TDFR (TSEC transmit deferral packet counter), 75
TDRP (TSEC drop frame counter register), 78
TEDF (TSEC transmit excessive deferral packet counter register), 75
TFCS (TSEC transmit FCS error counter register), 79
TFRG (TSEC transmit fragment counter register), 81
TIMING_CFG_0 (DDR SDRAM timing configuration register 0), 14
TIMING_CFG_1 (DDR SDRAM timing configuration register 1), 16
TIMING_CFG_2 (DDR SDRAM timing configuration register 2), 18
TIMING_CFG_3 (DDR SDRAM timing configuration register 3), 13
TJBR (TSEC jabber frame counter register), 79
TLCL (TSEC transmit late collision packet counter register), 77
TMCA (TSEC transmit multicast packet counter register), 73
TMCL (TSEC transmit multiple collision packet counter register), 76
TNCL (TSEC transmit total collision counter register), 78
TOVR (TSEC transmit oversize frame counter register), 80
TPKT (TSEC transmit packet counter register), 73
TR127 (TSEC transmit and receive 65- to 127-byte frame counter register), 61
TR1K (TSEC transmit and receive 512- to 1023-byte frame counter register), 62
TR255 (TSEC transmit and receive 128- to 255-byte frame counter register), 61
TR511 (TSEC transmit and receive 256- to 511-byte frame counter register), 62
TR64 (TSEC transmit and receive 64-byte frame counter register), 60
TRMAX (TSEC transmit and receive 1024- to 1518-byte frame counter register), 63
TRMGV (TSEC transmit and receive 1519- to 1522-byte VLAN frame counter register), 63
TSCL (TSEC transmit single collision packet counter register), 76
TSTAT (TSEC transmit status register), 34
TUND (TSEC transmit undersize frame counter register), 81
TXCF (TSEC transmit control frame counter register), 80
TXCL (TSEC excessive collision packet counter register), 77
TXFILLTUNING (USB transmit FIFO tuning controls register), 35
TXIC (TSEC transmit interrupt coalescing configuration register), 35
TXPF (TSEC transmit pause control frame counter register), 74
TXTTFILLTUNING (USB transaction translator transmit FIFO tuning controls register), 37

U

UAFR*n* (DUART alternate function registers 0–1), 16
UDLB*n* (DUART divisor least significant byte registers 0–1), 7
UDMB*n* (DUART divisor most significant byte registers 0–1), 7
UDSR*n* (DUART DMA status registers 0–1), 6, 17
UFCR*n* (DUART FIFO control registers 0–1), 10
UIER (DUART interrupt enable register), 8
UIIR*n* (DUART interrupt ID registers 0–1), 9
ULCR*n* (DUART line control registers 0–1), 11
ULPI VIEWPORT (ULPI register access), 37
ULSR*n* (DUART line status registers 0–1), 14
UMCR*n* (DUART MODEM control registers 0–1), 13
UMSR*n* (DUART MODEM status registers 0–1), 15
URBR*n* (DUART receiver buffer registers 0–1), 3, 6
USBCMD (USB command register), 23
USBINTR (USB interrupt enable register), 28
USBMODE (USB mode register), 47
USBSTS (USB status register), 26
USCR*n* (DUART scratch registers 0–1), 16
UTHR*n* (DUART transmitter holding registers 0–1), 3, 6

V

VID (vendor ID configuration register)

Index 2
General Index

A

Acronyms and abbreviated terms, 1-lxxxi
Address broadcast enable, 7-22
Address mask (LBC), 10-12
Address multiplexing (LBC SDRAM), 10-51
Advanced encryption standard execution units (AESU),
  see SEC, execution units, AESU
Alignment
  overview, 7-32
Arbiter, *see* CSB arbiter and bus monitor
Arbitration
  I^2C interface
    arbitration control, 17-14
    loss of arbitration—forcing of slave mode, 17-24
    procedure for arbitration, 17-14
ARC four execution unit (AFEU), *see* SEC, execution units,
    AFEU
Architecture, overview of device, 1-7
Architecture, PowerPC, 7-12

B

Battery backup, self-refresh mode, 9-69
Big-endian, 12-4, 12-23, 13-29, 16-18
Block address translation (BAT), 7-3
  see also Memory Management Unit (MMU), 7-3
Block diagram, e300 core, 7-2
Block diagrams
  clock subsystem, 4-29
  DDR controller, 9-2, 9-38
  delay lock loop (DLL), 22-1
  DMA/messaging unit, 12-1
    DMA controller, 12-20
  DUART, 18-2
  e300 core, 1-9
  general purpose timers, 5-52
  GPIO*n* module, 21-1
  I/O sequencer, 11-1
  I^2C interface, 17-1
  IPIC interupt sources, 8-3
  JTAG interface, 20-1
  local bus controller (LBC), 10-1
  MPC8349EA, 1-2
  PCI, 13-2
  periodic interval timer, 5-46, 5-51
  real time clock module, 5-38, 5-44
  SEC
    connection to internal bus, 14-3
    functional modules, 14-3
  security engine, 1-10
  SPI, 19-2
  timer pair-cascaded mode, 5-67
  timers super-cascaded mode, 5-67
  TSEC, 15-5
  USB interface, 16-2
  watchdog timer, 5-32, 5-36
Boot sequencer
  I^2C interface, 4-24–4-26, 17-2, 17-16–17-20
Boundary-scan testing, *see* JTAG interface
Branch processing unit (BPU), 7-1
  overview, 7-7
Branch trace enable (BE), 7-17
Breakpoints
  signaling, 7-37
Buffer descriptors, *see* TSEC, buffer descriptors
Bus interface
  I^2C, 1-15
  PCI bus arbitration unit, 1-12
  SEC, 14-10, 14-101
Bus interface unit (BIU), 7-10
Bus monitor, *see* CSB arbiter and bus monitor

C

Caches
  cache locking
    way locking, 7-29
  operations, 7-10
  way-locking, 7-29
$\overline{cint}$ (critical interrupt signal), 8-2
CLKIN signal, 4-3
Clock multiplier, 7-12
Clocks
  DDR clock distribution, 9-54
  I^2C
    clock stretching, 17-16
    clock synchronization, 17-15
    input synchronization and digital filter, 17-15
  introduction, 4-29
  LBC bus clocks and clock ratios, 10-3

clock ratio register (LCRR), 10-30
PCI agent mode, 4-30
PCI host mode, 4-30
PCI clock outputs (PCI_CLK_OUT[0:7]), 4-30
signals, 4-3
CLKIN, 4-3
PCI_CLK, 4-3
PCI_CLK_OUT[0:7], 4-3, 4-30
PCI_SYNC_IN, 4-3
PCI_SYNC_OUT, 4-3
subsystem block diagram, 4-29
system
domains, 4-30
registers
configuration, 4-37–4-41
TSEC
inputs and outputs, 15-10
management clock out (EC_MDC), 15-54
Coherent system bus (CSB)
arbiter, *see also* CSB arbiter and bus monitor
overview, 6-1
Completion unit, overview, 7-8
Condition register (CR), 7-15
Configuration
boot sequencer, 4-17
DDR, 9-10–9-30, 9-41
LBC
configuration register (LBCR), 10-29
SDRAM configurations supported, 10-48
PCI
host/agent mode, 13-4
PCI arbiter, 13-4
single 64-bit/dual 32-bit interface, 13-4
reset, 4-8
sampled signals, 3-19
see also Reset, configuration
TSEC interfaces, 15-126–15-145
Configurations
USB external ports, 16-3
Controller, *see* SEC, controller
Conventions
notational, 1-lxxx
signals, 1-lxxxi
Core interface
Core, *see* e300 core
CR (condition register)
overview, 7-15
Critical input (*cint*) interrupt, 7-32
Critical interrupt
exception enable (G2_LE only), 7-17
Crypto-channels, *see* SEC, crypto-channels
CSB arbiter and bus monitor
coherent system bus, 6-1
error handling sequence, 6-17
features, 6-1
functionality
arbitration policy, 6-11
address bus arbitration after ARTRY, 6-13
address bus arbitration with PRIORITY[0:1], 6-12
address bus arbitration with REPEAT, 6-13
address bus parking, 6-13
data bus arbitration, 6-13
bus error detection, 6-14
address only transaction type, 6-15
address time out, 6-14
data time out, 6-14
illegal (ECIWX/ECOWX) transaction type, 6-16
reserved transaction type, 6-15
transfer error, 6-14
initialization sequence, 6-16
memory map/register definition, 6-2
overview, 6-1
registers, 6-2–6-10

D

Data address translation, 7-17
Data cache enable, 7-21
Data cache flash invalidate, 7-22
Data cache lock, 7-21
Data cache way lock, 7-24
Data encryption standard execution unit (DEU),
see SEC, execution units, DEU
Data packet descriptors, *see* SEC, descriptors
Data TLB miss on load interrupt, 7-32
Data TLB miss on store interrupt, 7-32
DBAT, *see* Block address translation (BAT)
DDR controller
address signal mappings, 9-4
battery-backed RAM and forced self-refresh mode, 9-69
block diagram, 9-2, 9-38
clock distribution, 9-54
configuration, example, 9-41
data beat ordering, 9-61
driver impedance calibration, 9-8
error checking and correcting (ECC), 9-62
testing ECC with error injection, 9-31–9-32
error handling, 9-32, 9-64
features, 9-2
functional description, 9-38
initialization/application information, 9-65
programming different memory types, 9-66
interrupts, 9-35
memory map/register definition, 9-9
modes of operation, 9-3

on-die termination for CSs, 9-8
page mode and logical bank retention, 9-61
register descriptions, 9-10
by acronym, *see* Register Index
configuration registers, 9-10–9-30
error handling registers, 9-32–9-38
error injection registers, 9-31–9-32
SDRAM operation, 9-43
address multiplexing, 9-45
initialization sequence, 9-69
JEDEC standard interface commands, 9-49
mode-set command timing, 9-55
organizations supported, 9-43
refresh operation, 9-57
power-saving modes, 9-58
timing, 9-58
registered DIMM mode, 9-56
timing, 9-51
write timing adjustments, 9-56
self-refresh
operation in sleep mode, 9-60
using forced mode for battery backup, 9-69
signals summary, 9-3
see also Signals, DDR
DDR memory controller
debug configuration, 5-28
overview, 1-10
Debug configuration, 5-28
DDR, 5-28
local bus, 5-28
Debug facilities, 7-37
Debug modes
LBC source ID debug mode, 10-3
Decrementer, 7-32
Delay lock loop (DLL)
block diagram, 22-1
features, 22-2
initialization/application information, 22-2
memory map/register definition, 4-41, 22-3
modes of operation, 22-2
overview, 22-1
signals, 22-2
Distinctions
between MPC834x family products, 1-1
DMA controller, *see* DMA/messaging unit, DMA controller
DMA engine, 16-58
DMA/messaging unit
block diagram, 12-1
DMA controller
block diagram, 12-20
descriptors, 12-22
big endian mode, 12-23
DMA chain, 12-23
little endian mode, 12-24
halt and error conditions, 12-22
operation, 12-20
coherency, 12-22
overview, 1-16, 12-19
features, 12-2
functional description, 12-18
initialization steps
in chaining mode, 12-24
in direct mode, 12-24
memory map/register definition, 12-3
message unit, 12-18
doorbell registers, 12-19
messaging registers, 12-19
registers, 12-4–12-18
by acronym, *see* Register Index
configuration, control, and status registers, 12-4–12-11
Do-Complete-Split state
asynchronous, 16-104
periodic interrupt, 16-114
Doorbell registers, 12-8–12-9
Do-Start-Split state
asynchronous, 16-104
periodic interrupt, 16-113
DSI (data storage interrupt), 7-31
DTLB, 7-3
Dual universal asynchronous receiver/transmitters, *see* DUART
DUART
asynchronous communication bits, 18-2
parity bit, 18-19
START bit, 18-19
STOP bit, 18-20
baud-rate generator logic, 18-20
block diagram, 18-2
divisor latch access bit (ULCR*n*[DLAB]), 18-4, 18-11
error handling, 18-21
framing error, 18-9, 18-14, 18-19, 18-20, 18-21
overrun error, 18-21
parity error, 18-21
features, 18-2
functional description, 18-18
initialization/application information, 18-22
interrupt handling
interrupt control logic, 18-22
interrupt enable and control registers, 18-8–18-10
memory map/register definition, 18-4–18-5
modes of operation, 18-3
DMA mode selection, 18-22
FIFO mode, 18-21
interrupts, 18-21

local loop-back mode, 18-20
overview, 18-1
PC16450 UART compatibility, 18-2
registers, 18-6–18-18
by acronym, *see* Register Index
serial interface data format, 18-2
serial interface operation, 18-19–18-20
data transfer, 18-19
START bit, 18-19
STOP bit, 18-20
transaction protocol example, 18-19
signals, 18-3–18-4
UART_CTS[0:1] (DUART clear to send), 18-1, 18-3, 18-4
UART_RTS[0:1] (DUART request to send), 18-1, 18-3, 18-4
UART_SIN [0:1] (DUART transmitter serial data in), 18-3
UART_SOUT [0:1] (DUART transmitter serial data out), 18-3, 18-4
Dynamic power management enable, 7-20

E

e300 core
block diagram, 1-9
overview, 1-7
e300 core, differences between cores, 7-38
EC_GTX_CLK125 (TSEC gigabit transmit 125 MHz source) signals, 15-10
EC_MDC (TSEC management data clock) signals, 15-10
EC_MDIO (TSEC management data input/output) signals, 15-10
Error handling
CSB arbiter and bus monitor, 6-17
DDR, 9-32–9-38, 9-64
DMA/messaging unit, 12-22
DUART, 18-21
framing error, 18-9, 18-14, 18-19, 18-20, 18-21
overrun error, 18-21
parity error, 18-21
I^2C interface
boot sequencer mode, 4-27, 17-16
LBC
transfer error registers, 10-26–10-29
TSEC, 15-119–15-120
Ethernet controller, *see* TSEC
Exception little-endian mode, 7-16
Exception prefix, 7-17
Exceptions
overview, 7-29
Execution units (EU)
SEC, *see* SEC, execution units
External interrupt enable, 7-16

F

Features
overview of device features, 1-2
FIFO RAM controller, 16-58
FIFOs
AESU, 14-81
AFEU, 14-51
crypto-channels, 14-90
DEU, 14-42
MDEU, 14-63
RNG, 14-68
Floating-point available, 7-17
Floating-point exception mode 0, 7-17
Floating-point exception mode 1, 7-17
Floating-point model
FP registers (FPR*n*), 7-15
FPR*n* (floating-point registers 0–31), 7-15
Force branch indirect on bus, 7-22
FPR*n* (floating-point registers 0–31), 7-15
FPSCR (floating-point status and control reg.), 7-15
FSTNs
host controller operational model, 16-108
software operational model, 16-110

G

G2
overview, 7-12
General Purpose I/O module (GPIO), *see* GPIO
General purpose timers (GTM), 5-52
block diagram, 5-52
external signal description, 5-54
features, 5-53
functional description, 5-65
capture modes, 5-65
cascaded modes, 5-66
general-purpose timer units, 5-65
reference modes, 5-65
initialization/application information, 5-68
memory map/register definition, 5-56
modes of operation, 5-53
capture, 5-54
cascaded, 5-53
clock source, 5-54
reference, 5-54
overview, 5-52
registers, 5-57–5-64
GPCM (LBC general-purpose chip-select machine), 10-35
see also Local bus controller (LBC)
GPIO

block diagram, 21-1
features, 21-2
memory map/register definition, 21-2–21-3
overview, 21-1
registers, 21-3–21-6
signals, 21-2
GPR*n* (general-purpose registers 0–31), 7-15

H

Hash table algorithm, *see* TSEC, hash function
Header dword, 14-16
see also SEC, descriptors, header dword
HID*n* (hardware implementation registers 0–2)
PLL configuration, 7-23
High BAT enable, 7-24
HRESET, 4-7

I

I/O sequencer, 13-2
block diagram, 11-1
features, 11-2
functional description, 11-6
PCI outbound address translation, 11-7
transaction forwarding, 11-6
from the CSB port, 11-7
from the DMA port, 11-7
from the PCI ports, 11-7
transaction ordering, 11-8
memory map/register definition, 11-2–11-3
overview, 11-1
registers, 11-3–11-6
I^2C interface
arbitration
arbitration control, 17-14
loss of arbitration—forcing of slave mode, 17-24
procedure for arbitration, 17-14
block diagram, 17-1
boot sequencer mode, 4-24–4-26, 17-2, 17-16–17-20
error condition behavior, 4-27, 17-16
calling address match condition, 17-5
clock control, 17-15
clock stretching, 17-16
clock synchronization, 17-15
input synchronization and digital filter, 17-15
master mode, 17-15
slave mode, 17-15
data transfer, 17-12
error handling
boot sequencer mode, 4-27, 17-16
features, 17-2
frequency divider
frequency divider register (I2CFDR), 17-5
functional description, 17-10
handshaking, 17-15
implementation details, 17-12
address compare, 17-14
control transfer, 17-13
transaction monitoring, 17-13
initialization/application information, 17-20–17-21
boot sequencer mode, *see* I^2C interface, boot sequencer mode
generation of SCL when SDA low, 17-23
initialization sequence, 17-22
post-transfer software response, 17-22
repeated START generation, 17-23
START generation, 17-11, 17-22
STOP generation, 17-12, 17-23
interrupts
calling address match condition, 17-5
flowchart for interrupt service routine, 17-20
interrupt after transfer, 17-22
interrupt enable bit (I2CCR[MIEN]), 17-7
interrupt on START, 17-22
interrupt pending status bit (I2CSR[MIF]), 17-9
interrupt-driven byte-to-byte transfers, 17-2
read of last byte, 17-23
slave mode interrupt service routine guidelines, 17-23
for slave transmitter routine, 17-24
loss of arbitration, 17-24
memory map/register definition, 17-4
modes of operation, 17-2
boot sequencer mode, 17-2, 17-16–17-20
interrupt-driven byte-to-byte data transfer, 17-2
master mode, 17-2
slave mode, 17-2
overview, 1-15
register descriptions, 17-5–17-10
by acronym, *see* Register Index
serial data/clock wires, 17-1
signals, 17-3–17-4
see also Signals, I^2C
transaction protocol, 17-10
handshaking, 17-15
repeated START condition, 17-3, 17-12
slave address transmission, 17-11
START condition, 17-3, 17-11, 17-22
STOP condition, 17-3, 17-12, 17-23
IBAT*n*U/L (instruction block address translation regs. 0–7, upper/lower), 7-3
IEEE 1149.1 specifications
specification compliance, 20-3
Initialization
DDR (initialization and application information), 9-65

programming different memory types, 9-66
I2C interface (initialization and application information)
STOP generation, 17-23
Initialization/application information
CSB arbiter and bus monitor, 6-16
delay lock loop (DLL), 22-2
DMA/messaging unit, 12-24
GTM registers, 5-68
host controller, 16-80
I2C interface, 17-20–17-21
boot sequencer mode, *see* I2C interface, boot sequencer mode
generation of SCL when SDA low, 17-23
initialization sequence, 17-22
post-transfer software response, 17-22
repeated START generation, 17-23
START generation, 17-11, 17-22
STOP generation, 17-12
LBC, 10-80
LBC SDRAM power-on initialization, 10-49
PCI, 13-61
power management control, 5-74
real time clock module
RTC programming guidelines, 5-45
SPI, 19-16
master programming example, 19-16
slave programming example, 19-16
TSEC, 15-126–15-145
watchdog timer, 5-38
Instruction address translation, 7-17
Instruction cache enable, 7-21
Instruction cache flash invalidate, 7-21
Instruction cache lock., 7-21
Instruction cache way lock, 7-24
Instruction timing
overview, 7-34–7-35
see also Execution timing
$\overline{int}$ (internal interrupt signal), 8-2
INTA, 12-4, 12-19
Integrated programmable interrupt controller, *see* IPIC
Intel PC133 SDRAM commands (LBC), 10-50
Interfaces
I2C, 1-15
JTAG
block diagram, 20-1
TAP controller, 20-3
PCI interface
bus arbitration unit, 1-12
USB, 16-2
Interrupts
crypto-channel, 14-92
see also SEC, crypto-channels, interrupts
DDR, 9-35
DUART
interrupt control logic, 18-22
interrupt enable and control registers, 18-8–18-10
I2C interface
calling address match condition, 17-5
flowchart for interrupt service routine, 17-20
interrupt after transfer, 17-22
interrupt enable bit (I2CCR[MIEN]), 17-7
interrupt on START, 17-22
interrupt pending status bit (I2CSR[MIF]), 17-9
interrupt-driven byte-to-byte transfers, 17-2
read of last byte, 17-23
slave mode interrupt service routine guidelines, 17-23
for slave transmitter routine, 17-24
loss of arbitration, 17-24
LBC interrupt register, 10-27
SEC, 14-103
split transaction, 16-105
TSEC, 15-116–15-118
interrupt registers, 15-19–15-25
IPIC
block diagram, 8-3
features, 8-4
functional description, 8-28
interrupts
configuration, 8-29
highest priority, 8-31
internal
group relative priority, 8-30
machine check, 8-35
masking sources, 8-34
mixed
group relative priority, 8-30
request masking, 8-35
source priorities, 8-31
levels, 8-31
types, 8-28
vector generation and calculation, 8-35
memory map/register definition, 8-6–8-7
modes of operation, 8-4
core disable mode, 8-5
core enable mode, 8-4
overview, 8-1
registers, 8-8–8-28
by acronym, *see* Register Index
signals, 8-6
$\overline{cint}$ (critical interrupt), 8-2
$\overline{int}$ (internal interrupt), 8-2
$\overline{mcp}$ (machine check processor), 8-2
overview, 8-5
$\overline{smi}$ (system management interrupt), 8-2

ISI (instruction storage interrupt), 7-31
Isochronous transfer descriptor (iTD), *see* USB interface, isochronous (high-speed) transfer descriptor (iTD)

J

JEDEC SDRAM commands (LBC), 10-50
Joint test action group, *see* JTAG interface
JTAG interface
  block diagram, 20-1
  registers, 20-3
  signals, 20-1–20-3
  TAP (test access port) controller, 20-3
  TAP controller, 20-3
JTAG test and debug interface, 7-12, 7-37

L

LA[27:31] (LBC non-multiplexed address) signals, 10-7
LAD[0:31] (LBC multiplexed address/data) signals, 10-8
LALE (LBC external address latch enable) signal, 10-5, 10-33
LBC, *see* Local bus controller (LBC)
LBCTL (LBC data buffer control) signal, 10-7, 10-35
LBS[0:3] (LBC UPM byte select) signals, 10-6
LCK[0:2] (LBC clock) signals, 10-8
LCKE (LBC clock enable) signal, 10-8
LCS[0:7] (LBC chip select) signals, 10-5
LCS0 (LBC chip select 0) signal, 10-47
LDP[0:3] (LBC data parity) signals, 10-8, 10-35
LGPL0 (LBC GP line 0) signal, 10-6
LGPL1 (LBC GP line 1) signal, 10-6
LGPL2 (LBC GP line 2) signal, 10-6
LGPL3 (LBC GP line 3) signal, 10-6
LGPL4 (LBC GP line 4) signal, 10-7
LGPL5 (LBC GP line 5) signal, 10-7
LGTA (LBC GPCM transfer acknowledge) signal, 10-7, 10-46
Link table, *see* SEC, descriptors, link table
Little-endian, 16-18
Little-endian mode enable, 7-18
Load/store unit (LSU), 7-1
  overview, 7-8
Local bus controller (LBC)
  address and address space checking, 10-32
  address mask field—option registers, 10-12
  block diagram, 10-1
  boot chip-select operation, 10-47
  bus monitor, 10-35
  bus turnaround, 10-83
    additional address phases (UPM cycles), 10-84
    address following read, 10-83
    read data following address, 10-83
    read-modify-write cycle (parity), 10-84
  clocks and clock ratios, 10-3
    clock ratio register (LCRR), 10-30
  configuration
    LBC configuration register (LBCR), 10-29
  DSP hosts (interface to), 10-95
    MSC8102 DSI interface, 10-95
  error handling
    transfer error registers, 10-26–10-29
  external access termination (LGTA), 10-46
  features, 10-2
  functional description, 10-31
  general-purpose chip-select machine (GPCM), 10-35
    chip-select and write enable negation timing, 10-41
    chip-select assertion timing, 10-40
    extended hold time on read accesses, 10-45
    GPCM mode
      registers, 10-13
    output enable timing, 10-45
    programmable wait state configuration, 10-41
    relaxed timing, 10-42
    timing configuration, 10-37
  initialization/application information, 10-80
  interrupts
    transfer error interrupt enable register (LTEIR), 10-27
  memory map/register definition, 10-9
  memory refresh timer prescaler, 10-21
  modes of operation, 10-3
    bus clock and clock ratios, 10-3
    GPCM mode, registers, 10-13
    SDRAM mode, registers, 10-17
    source ID debug mode, 10-3
    UPM mode, registers, 10-16
  overview, 1-14
  parity generation and checking, 10-35, 10-93
  peripherals, 10-80
    GPCM timing, 10-82
    hierarchy for very high speeds, 10-81
    hierarchy on the local bus, 10-81
    multiplexed address/data, 10-80
  port sizes, 10-84
  register descriptions, 10-10
    by acronym, *see* Register Index
  SDRAM interface, 10-48–10-58, 10-86
    address multiplexing, 10-51
    basic capabilities, 10-86
    commands (Intel PC133 and JEDEC), 10-50
    configurations supported, 10-48
    device-specific parameters, 10-52
    limitations, 10-88
    maximum SDRAM supported, 10-87
    page hit checking, 10-51

page management, 10-51
parity support, 10-93
power-on initialization, 10-49
refresh, 10-58
SDRAM mode
registers, 10-17, 10-22
timing, 10-55
activate-to-read/write interval, 10-53
CAS latency, 10-53
external buffers, 10-55
MODE-SET commands, 10-58
precharge-to-activate interval, 10-52
refresh recovery, 10-54
refresh timing, 10-58
write recovery, 10-54
transactions, 10-57
signals, 10-4
see also Signals, LBC
UPM interfaces, 10-59–10-79
block diagram, 10-59
example interface, 10-74
extended hold time (reads), 10-74
programming the UPMs, 10-62
RAM array, 10-65
address multiplexing, 10-71
byte select signal timing, 10-69
chip select signal timing, 10-69
data timing, 10-72
general purpose signal timing, 10-70
LGPL[0:5] timing (LAST), 10-73
loop control, 10-70
RAM word definition, 10-66
REDO, 10-71
wait mechanism (WAEN), 10-73
signal timing, 10-64
synchronous UPWAIT (early transfer acknowledge), 10-74
UPM mode
registers, 10-16, 10-18
UPM requests, 10-60
exception requests, 10-62
memory access requests, 10-61
refresh timer requests, 10-61
software requests, 10-62
ZBT SRAM interface, 10-94
LOE (LBC GPCM output enable) signal, 10-6
LPBSE (LBC parity byte select) signal, 10-7
LSDA10 (LBC SDRAM A10) signal, 10-6
LSDCAS (LBC SDRAM CAS) signal, 10-6
LSDDQM[0:3] (LBC SDRAM data mask) signal, 10-6
LSDRAS (LBC SDRAM RAS) signal, 10-6
LSDWE (LBC SDRAM write enable) signal, 10-6
LSYNC_IN (LBC DLL synchronization in) signal, 10-8
LSYNC_OUT (LBC DLL synchronization out) signal, 10-8
LWE[0:3] (LBC GPCM write enable) signals, 10-6

M

MA[0:14] (DDR address bus) signals, 9-6
MAC functionality
see TSEC, MAC functionality
Machine check enable, 7-17
MBA[0:1] (DDR logical bank address) signals, 9-6
MCAS (DDR column address strobe) signal, 9-7
MCK[0:5] (DDR clock output complement) signals, 9-8
MCK[0:5] (DDR clock output) signals, 9-8
MCKE[0:3] (DDR clock enable) signals, 9-8
mcp (machine check processor signal), 8-2
MCS[0:3] (DDR chip select) signals, 9-7
MDIC[0:1] (DDR driver impedance calibration) signals, 9-8
MDM[0:8] (DDR SDRAM data output mask) signals, 9-8
MDQ[0:8] (DDR data bus strobe) signals, 9-5, 9-40
MDVAL (DDR/LBC debug mode data valid) signal, 10-9
MECC[0:7] (DDR error correcting code) signals, 9-6
Memory accesses, 7-36
Memory management unit (MMU)
overview, 7-3, 7-9, 7-33
Memory maps
accessing CCSR memory from external masters, 5-16
address translation and mapping, 5-3
complete IMMR map, 2-1
configuring local access windows, 5-14
CSB arbiter and bus monitor, 6-2
DDR controller, 9-9
delay lock loop (DLL), 4-41, 22-3
distinguishing local access windows from other mapping functions, 5-14
DMA/messaging unit, 12-3
DUART, 18-4–18-5
general purpose timers, 5-56
GPIO, 21-2–21-3
I/O sequencer, 11-2–11-3
I^2C, 17-4
inbound address translation and mapping windows, 5-15
IPIC, 8-6–8-7
LBC, 10-9
local access register, 5-4
local access windows
intoduction, 5-4
precedence, 5-14
outbound address translation and mapping windows, 5-15
PCI, 13-14–13-16
periodic interval timer, 5-47
power management control, 5-69
real time clock module, 5-40

reset, 4-32
SEC, 14-10, 14-11–14-15
SPI, 19-8
TSEC
detailed, 15-13–15-18
top-level, 15-12
USB interface, 2-18–2-20, 16-16–16-18
watchdog timer, 5-33
window into configuration space, 5-3
Message digest execution unit (MDEU), *see* SEC, execution units, MDEU
MODT[0:3] (DDR on-die termination) signals, 9-8
MPC603e core, *see* e300 core
MPC8343E
functionality, 1-1
MPC8347E
functionality, 1-1
MPC834x family
product distinctions, 1-1
MRAS (DDR row address strobe) signal, 9-7
MSR (machine state register), 7-16
MSRCID[0:4] (DDR/LBC debug source ID) signals, 10-9
MWE (DDR write enable) signal, 9-7

N

No-op the data cache touch instructions, 7-22

P

Page hit checking (LBC SDRAM), 10-51
Page management (LBC SDRAM), 10-51
Parameter memories, *see* SEC, execution units, PKEU, parameter memories
PCI
64-bit/32-bit bus, 13-4
address map
address translation
PCI outbound, 11-7
block diagram, 13-2
bridge
arbitration example, 13-46
PCI parity operation, 13-59
bus arbitration, 13-45
alogrithm, 13-45
broken master lock-out, 13-46
master latency timer, 13-47
parking, 13-45
bus arbitration unit, 1-12
bus commands, 13-47
bus error functions, 13-58
parity, 13-58
reporting, 13-58
bus operations, 13-55
agent mode configuration access, 13-57
data streaming, 13-55
dual address cycles, 13-55
fast back-to-back transactions, 13-55
host mode configuration access, 13-56
interrupt acknowledge, 13-58
special cycle command, 13-57
CompactPCI Hot Swap specification support, 13-61
DMA controller, *see* DMA/messaging unit, DMA controller
features, 13-3
functional description, 13-45
inbound address translation, 13-60
inbound windows, 5-15
initialization/application information, 13-61
sequence for agent mode, 13-61
sequence for host mode, 13-61
memory map/register definition, 13-14–13-16
modes of operation, 13-3
host/agent mode configuration, 13-4
PCI arbiter configuration, 13-4
single 64-bit/dual 32-bit interface configuration, 13-4
output hold configuration, 4-21
overview, 1-12
protocol fundamentals, 13-48
addressing, 13-48
basic transfer control, 13-48
bus driving and turnaround, 13-50
bus transactions, 13-50
byte enable signals, 13-49
device selection, 13-49
read and write transactions, 13-50
burst read example, 13-51
burst write example, 13-52
single beat read example, 13-51
single beat write example, 13-52
transaction termination, 13-52
target-initiated terminations, 13-53
registers, 13-16–13-44
configuration access, 13-16–13-18
configuration space, 13-29–13-44
base class code, 13-35
BIST control, 13-37
cache line size, 13-36
capabilities pointer, 13-40
device ID, 13-31
GPL base address register 0, 13-38
GPL base address registers 1,2, 13-38
GPL extended base address registers 1,2, 13-39
header type, 13-37
hot swap register block, 13-44

interrupt line, 13-41
interrupt pin, 13-41
latency timer, 13-36
MAX LAT, 13-42
MIN GNT, 13-41
PCI arbiter control, 13-43
PCI command, 13-32
PCI function, 13-42
PCI status, 13-33
PIMMR base address, 13-37
revision ID, 13-34
standard programming interface, 13-34
subclass code, 13-35
sub-system device ID, 13-40
sub-system vendor ID, 13-39
vendor ID, 13-31
control and status, 13-18–13-29
signals, 13-4–13-14
PCI_C/BE[7:0] (PCI command/byte enable) signals, 13-8
PCI_CLK, 4-3
PCI_CLK_OUT[0:7], 4-3, 4-30
PCI_PAR64 (PCI upper DWORD parity) signal, 13-11
PCI_PERR (PCI parity error) signal, 13-11
PCI_REQ[4:0] (PCI bus request) signals, 13-11, 13-12
PCI_REQ64 (PCI 64-bit transaction request) signal, 13-12
PCI_SERR (PCI system error) signal, 13-13
PCI_STOP (PCI stop) signal, 13-13
PCI_SYNC_IN, 4-3
PCI_SYNC_OUT, 4-3
PCI_TRDY (PCI target ready) signal, 13-14
Periodic interval timer (PIT), 5-45
block diagram, 5-46
external signal description, 5-46
features, 5-46
functional block diagram, 5-51
functional description, 5-50
memory map/register definition, 5-47
modes of operation, 5-46
operational modes, 5-51
overview, 5-45
registers, 5-47–5-50
PHY clocks, 16-15
PHY interface, 16-58
PIC, *see* IPIC
Pointer dwords, *see* SEC, descriptors, pointer dwords
Power management
DDR interface, 9-58
modes, 7-11
Power management control, 5-68
external signal description, 5-69
functional description, 5-71
dynamic power management, 5-71
exiting low power states, 5-73
shutting down unused blocks, 5-72
software-controlled power-down states, 5-72
initialization/application information, 5-74
memory map/register definition, 5-69
modes of operation, 5-68
registers, 5-69–5-71
Power saving mode
SEC, 14-103
Power-on reset (POR)
bidirectional signal states during reset, 3-21
flow, 4-5
output signal states during reset, 3-19
reset configuration signals, 3-19
timing diagram, 4-7
PowerPC architecture
levels of implementation, 7-12
overview, 7-12
Privilege level (PR), 7-17
Processor core, *see* e300 processor core
Program interrupt, 7-32
Programmable interrupt controller, *see* IPIC

Q

Queue element transfer descriptor (qTD), *see* USB interface, queue element transfer descriptor (qTD)
Queue heads, *see* USB interface, queue heads

R

Random number generator (RNG), *see* SEC, execution units, RNG
Real time clock module (RTC), 5-38
block diagram, 5-38, 5-44
external signals description, 5-39
features, 5-39
functional description, 5-43
initialization/application information
RTC programming guidelines, 5-45
memory map/register definition, 5-40
modes of operation, 5-39
operational modes, 5-44
overview, 5-38
registers, 5-40–5-43
Recoverable exception, 7-18
Registers
by acronym, *see* Register Index
clock
configuration, 4-37–4-41
configuration registers
hardware implementation registers (HID*n*)
PLL configuration, 7-23

CSB arbiter and bus monitor, 6-2–6-10
DDR
configuration registers, 9-10–9-30
error handling registers, 9-32–9-38
error injection registers, 9-31–9-32
DMA/messaging unit, 12-4–12-18
configuration, control, and status registers, 12-4–12-11
DUART, 18-6–18-18
GPIO, 21-3–21-6
I/O sequencer, 11-3–11-6
I^2C interface, 17-5
IPIC, 8-8–8-28
JTAG
boundary-scan registers, 20-3
JTAG interface, 20-3
bypass register, 20-3
instruction register, 20-3
status register, 20-3
LBC, 10-10
PCI, 13-16–13-44
configuration access, 13-16–13-18
configuration space, 13-29–13-44
control and status, 13-18–13-29
reset
configuration, 4-11, 4-32–4-37
SEC
AESU, 14-68–14-81
controller, 14-93–14-99
crypto-channel, 14-82–14-90
DEU, 14-35–14-42
MDEU, 14-51–14-62
PKEU, 14-26–14-34
RNG, 14-63–14-68
SPI, 19-9–19-16
time base facility (TBL/TBU)
for reading, 7-16
TSEC
attribute registers, 15-89
FIFO control and status registers, 15-30–15-32
general control and status registers, 15-19–15-30
hash function registers, 15-87–15-89
MAC registers, 15-45–15-60
MIB registers, 15-60–15-87
receive control and status registers, 15-40–15-45
ten-bit interface (TBI) registers, 15-90–15-101
transmit control and status registers, 15-33–15-40
USB interface, 16-18–16-57
capability registers, 16-18–16-23
operational registers, 16-23–16-57
Reset
actions, 4-4
causes, 4-4
configuration, 4-8
boot memory space, 4-16
boot ROM location, 4-17
boot sequencer, 4-17
CLKIN division, 4-10
default words, 4-27
hard coded reset configuration word high, 4-28
hard coded reset configuration word low, 4-27
hard coded reset configuration words usage examples, 4-29
e300 core true little endian mode, 4-20
LALE, 4-21
LDP, 4-21
loading from I^2C EEPROM, 4-24
boot sequencer, 4-24
calling address, 4-24
data format in reset configuration mode, 4-24
load fail, 4-27
loading words, 4-21
from local bus EEPROM, 4-21
local bus EEPROM timing, 4-23
PCI host/agent, 4-16
selecting input signals, 4-10
signals, 4-9
TSEC1 mode, 4-19
TSEC2 mode, 4-20
words, 4-11
words source, 4-9
DDR debug configuration, 5-28
functional device description, 5-35, 5-65, 5-71
hard reset ($\overline{\text{HRESET}}$)
flow, 4-7
memory map/register definition, 4-32
operations, 4-3
PCI output hold configuration, 4-21
power-on reset (POR)
flow, 4-5
timing diagram, 4-7
registers
configuration, 4-11, 4-32–4-37
signals, 4-1–4-2
soft reset ($\overline{\text{SRESET}}$)
flow, 4-8
Reset and clocking blocks
boot ROM location, 4-17
boot sequencer configuration, 4-17
CPU boot configuration, 4-15
external signal description, 5-46, 5-54, 5-69
functional description, 4-3, 5-35, 5-65, 5-71
host/agent configuration, 4-16
memory map/register definition, 5-33, 5-39, 5-47, 5-56
system PLL ratio, 4-30

TSEC width, 4-19, 4-20

S

SCL (I^2C serial clock) signal, 17-3, 17-4
SDA (I^2C serial data) signal, 17-3, 17-4
SDRAM interface (LBC), 10-48–10-58
  see also Local bus controller (LBC), SDRAM interface
SEC
  architecture overview, 14-2
  block diagrams
    connection to internal bus, 14-3
    functional modules, 14-3
  bus interface, 14-10, 14-101
    arbitration priority, 14-102
    bus access, 14-101
      master read, 14-101
      master write, 14-102
      slave access, 14-102
    interrupts, 14-103
    snooping by caches, 14-103
  controller, 14-9, 14-93
    access, 14-9, 14-100
      host-controlled, 14-9
      multiple channels, 14-100
      multiple EU assignment, 14-100
      priority arbitration, 14-100
      snapshot arbiters, 14-101
    registers, 14-93–14-99
  crypto-channels, 14-8, 14-81
    descriptor buffer, 14-91
    fetch FIFO, 14-90
    interrupts, 14-92
      channel done, 14-92
      channel error, 14-92
      channel reset, 14-93
    registers, 14-82–14-90
  data packet descriptors, 14-4
  descriptors
    format, 14-16
    header dword, 14-16
      selecting descriptor type, 14-18
      selecting execution units, 14-17
    link table
      example, 14-23
      format, 14-21
    overview, 14-15
    pointer dwords, 14-19
    structure, 14-15
  execution units, 14-5, 14-26
    AESU, 14-7, 14-68
      FIFOs, 14-81
      registers, 14-68–14-81
    AFEU, 14-6, 14-42
      context memory, 14-50
      dump context, 14-43
      FIFOs, 14-51
      host-provided context, 14-43
    DEU, 14-6, 14-34
      FIFOs, 14-42
      registers, 14-35–14-42
    MDEU, 14-7, 14-51
      FIFO, 14-63
      registers, 14-51–14-62
    PKEU, 14-5, 14-26
      elliptic curve operations, 14-5
      modular exponentiation operations, 14-6
      parameter memories, 14-34
      registers, 14-26–14-34
    RNG, 14-7, 14-63
      FIFO, 14-68
      registers, 14-63–14-68
  features, 14-1
  power saving mode, 14-103
Security engine
  block diagram, 1-10
  overview, 1-10
Security engine (SEC)
  descriptors
    header dword
      writeback format, 14-84
Security engine, *see* SEC
Segment registers (SR*n*), 7-18
Serial data/clock wires, 17-1
Serial peripheral interface, *see* SPI
Signals
  alphabetical signal listing
    reference, 3-11–3-19
  bidirectional, active during reset, 3-21
  clock, 4-3
    CLKIN, 4-3
    PCI_CLK, 4-3
    PCI_CLK_OUT[0:7], 4-3, 4-30
    PCI_SYNC_IN, 4-3
    PCI_SYNC_OUT[0:7], 4-3
  complete signal listing
    reference by functional block, 3-4–3-11, 3-11–3-19
  configuration, sampled at reset, 3-19
  control, description, 5-39, 5-47, 5-55, 5-69
  conventions, 1-lxxxi
  DDR
    MA[0:14] (address bus), 9-6
    MBA[0:1] (logical bank address), 9-6
    MCAS (column address strobe), 9-7
    MCK[0:5] (DDR clock output complements), 9-8

MCK[0:5] (DDR clock outputs), 9-8
MCKE[0:3] (DDR clock enables), 9-8
MCS[0:3] (chip selects), 9-7
MDIC[0:1] (driver impedance calibration), 9-8
MDM[0:8] (SDRAM data output mask), 9-8
MDQS[0:8] (data bus strobes), 9-5, 9-40
MECC[0:7] (error correcting code), 9-6
MODT[0:3] (on-die termination), 9-8
MRAS (row address strobe), 9-7
MWE (write enable), 9-7
delay lock loop (DLL), 22-2
DUART, 18-3–18-4
UART_CTS[0:1] (DUART clear to send), 18-1, 18-3, 18-4
UART_RTS[0:1] (DUART request to send), 18-1, 18-3, 18-4
UART_SIN [0:1] (DUART transmitter serial data in), 18-3
UART_SOUT [0:1] (DUART transmitter serial data out), 18-3, 18-4
eset and clocking blocks, 5-46, 5-69
GPIO, 21-2
groupings figure, 3-1
I^2C
SCL (serial clock), 17-3, 17-4
SDA (serial data), 17-3, 17-4
IPIC, 8-6
cint (critical interrupt), 8-2
int (internal interrupt), 8-2
mcp (machine check processor), 8-2
smi (system management interrupt), 8-2
JTAG interface, 20-1–20-3
LBC
LA[27:31] (non-multiplexed address), 10-7
LAD[0:31] (multiplexed address/data), 10-8
LALE (external address latch enable), 10-5, 10-33
LBCTL (data buffer control), 10-7, 10-35
LBS[0:3] (UPM byte select), 10-6
LCK[0:2] (clock), 10-8
LCKE (clock enable), 10-8
LCS[0:7] (chip select), 10-5
LCS0 (LBC chip select 0), 10-47
LDP[0:3] (data parity), 10-8, 10-35
LGPL0 (GP line 0), 10-6
LGPL1 (GP line 1), 10-6
LGPL2 (GP line 2), 10-6
LGPL3 (GP line 3), 10-6
LGPL4 (GP line 4), 10-7
LGPL5 (GP line 5), 10-7
LGTA (GPCM transfer acknowledge), 10-7, 10-46
LOE (GPCM output enable), 10-6
LPBSE (parity byte select), 10-7
LSDA10 (SDRAM A10), 10-6
LSDCAS (SDRAM CAS), 10-6
LSDDQM[0:3] (SDRAM data mask), 10-6
LSDRAS (SDRAM RAS), 10-6
LSDWE (SDRAM write enable), 10-6
LSYNC_IN (DLL synchronization in), 10-8
LSYNC_OUT (DLL synchronization out), 10-8
LWE[0:3] (GPCM write enable), 10-6
MDVAL (debug mode data valid), 10-9
MSRCID[0:4] (debug source ID), 10-9
TA (data transfer acknowledge), 10-34
UPWAIT (UPM wait), 10-7, 10-60
muxing, 3-21
output, states during reset, 3-19
overview, 3-1, 7-36
PCI, 13-4–13-14
INTA, 12-4, 12-19
PCI_C/BE[7:0] (command/byte enable), 13-8
PCI_PAR64 (upper DWORD parity), 13-11
PCI_PERR (parity error), 13-11
PCI_REQ[4:0] (bus request), 13-11, 13-12
PCI_REQ64[4:0] (64-bit transaction request), 13-12
PCI_SERR (system error), 13-13
PCI_STOP (stop), 13-13
PCI_TRDY (target ready), 13-14
reset, 4-1–4-2
configuration, 4-9
reset and clocking blocks, 5-54
TSEC
EC_GTX_CLK125 (TSEC gigabit transmit 125 MHz source), 15-10
EC_MDC (TSEC management data clock), 15-10
EC_MDIO (TSEC management data input/output), 15-10
TSEC*n*_COL (TSEC 1–2 collision input), 15-9
TSEC*n*_CRS (TSEC 1–2 carrier sense input), 15-9
TSEC*n*_GTX_CLK (TSEC 1–2 gigabit transmit clock), 15-10
TSEC*n*_RX_CLK (TSEC 1–2 receive clock), 15-10
TSEC*n*_RX_DV (TSEC 1–2 receive data valid), 15-10
TSEC*n*_RX_ER (TSEC 1–2 receive error), 15-11
TSEC*n*_RXD[7:0] (TSEC 1–2 receive data in), 15-11
TSEC*n*_TX_CLK (TSEC 1–2 transmit clock in), 15-11
TSEC*n*_TX_EN (TSEC 1–2 transmit data valid in), 15-12
TSEC*n*_TX_ER (TSEC 1–2 transmit error in), 15-12
TSEC*n*_TXD[7:0] (TSEC 1–2 transmit data out), 15-11
USB interface, 16-4–16-15
see also USB interface, signals
Single-step
trace enable (SE), 7-17
smi (system management interrupt signal), 8-2

Soft reset and reconfiguring for TSEC, 15-110
Software watchdog timer, *see* Watchdog timer (WDT)
SPI, 1-17
  block diagram, 19-2
  features, 19-2
  initialization/application information, 19-16
    master programming example, 19-16
    slave programming example, 19-16
  memory map/register definition, 19-8
  modes of operation, 19-3
    master, 19-3
    multiple-master, 19-5
    slave, 19-4
  registers, 19-9–19-16
    by acronym, *see* Register Index
  signals, 19-7–19-8
    descriptions, 19-7
    overview, 19-7
  transmission and reception process, 19-3
Split transaction isochronous transfer descriptor (siTD), *see* USB interface, split transaction isochronous transfer descriptor (siTD)
Split transactions, *see* USB interface, split transactions
SPRG0–SPRG7, 7-3
SPRs (special purpose registers), 7-15–7-19
$\overline{\text{SRESET}}$, 4-8
SR*n* (segment registers 0–15), 7-18
Supervisor-level SPRs, 7-18
System call (**sc**), 7-32
System configuration
  local memory map overview, 5-1
  overview, 5-1
  registers, 5-17–5-28
System interface, 16-58
System management interrupt ($\overline{smi}$), 7-33
System timers
  overview, 1-17

T

TA (LBC data transfer acknowledge) signal, 10-34
TAP (test access port) controller, 20-3
TBL/TBU (time base facility)
  for reading, 7-16
  time base/decrementer, 7-11
Three-speed Ethernet controller, *see* TSEC
True little-endian, 7-23
TSEC
  attribute
    registers, 15-89
  block diagram, 15-5
  buffer descriptors, 15-120
    receive buffer descriptor (RxBD), 15-124
    transmit data buffer descriptor (TxBD), 15-122
  clocks
    inputs and outputs, 15-10
    management clock out (EC_MDC), 15-54
  configuration of interfaces, 15-126
    GMII interface mode, 15-130
    MII interface mode, 15-126
    RGMII interface mode, 15-137
    RTBI interface mode, 15-141
    TBI interface mode, 15-133
  error handling, 15-119–15-120
  features, 15-7
  functional description, 15-101
  gigabit Ethernet channel operation, 15-109
    flow control, 15-115
    frame reception, 15-112
    frame recognition, 15-113
      destination address recognition, 15-114
    frame transmission, 15-111
    initialization sequence, 15-109
      hardware controlled initialization, 15-109
      user initialization, 15-109
    inter-packet gap time, 15-119
    interrupt handling, 15-116–15-118
  hash function
    hash table algorithm, 15-115
    hash table effectiveness, 15-115
    registers, 15-87, 15-89
  initialization/application information, 15-126–15-145
  interrupt handling, 15-116
    interrupt registers, 15-19–15-25
  interrupts
    interrupt coalescing, 15-117
      by frame count threshold, 15-117
      by timer threshold, 15-118
  MAC functionality, 15-45–15-60
    configuration, 15-45
    CSMA/CD controlling, 15-45
    packet collisions, 15-46
    packet flow, 15-46
    PHY link control, 15-47
    registers, 15-47
  memory map/register definition, 15-12
    detailed, 15-13–15-18
    top level, 15-12
    TSEC2 controller offsets, 15-18
  modes of operation, 15-8
    10 Mbps and 100 Mbps MII operation, 15-8
    1000 Mbps GMII and TBI operation, 15-8
    address recognition options, 15-8
    full and half-duplex operation, 15-8
    internal and external loop back, 15-119

RMON support, 15-113
overview, 1-11, 15-6
physical interface connections, 15-101
gigabit media-independent interface (GMII), 15-102
media-independent interface (MII), 15-102
reduced gigabit media-independent interface (RGMII), 15-103
reduced ten-bit interface (RTBI), 15-105
ten-bit interface (TBI), 15-104
register descriptions, 15-19
attribute registers, 15-89
by acronym, *see* Register Index, 15-19
FIFO control and status registers, 15-30–15-32
general control and status registers, 15-19–15-30
hash function registers, 15-87–15-89
MAC registers, 15-45–15-60
MIB registers, 15-60–15-87
receive control and status registers, 15-40–15-45
ten-bit interface (TBI) registers, 15-90–15-101
transmit control and status registers, 15-33–15-40
signals, 15-9–15-12
see also Signals, TSEC
soft reset and reconfiguring procedure, 15-110
TSEC*n*_COL (TSEC 1–2 collision input) signals, 15-9
TSEC*n*_CRS (TSEC 1–2 carrier sense input) signals, 15-9
TSEC*n*_GTX_CLK (TSEC 1–2 gigabit transmit clock) signals, 15-10
TSEC*n*_RX_CLK (TSEC 1–2 receive clock) signals, 15-10
TSEC*n*_RX_DV (TSEC 1–2 receive data valid) signals, 15-10
TSEC*n*_RX_ER (TSEC 1–2 receive error) signals, 15-11
TSEC*n*_RXD[7:0] (TSEC 1–2 receive data in) signals, 15-11
TSEC*n*_TX_CLK (TSEC 1–2 transmit clock in) signals, 15-11
TSEC*n*_TX_EN (TSEC 1–2 transmit data valid in) signals, 15-12
TSEC*n*_TX_ER (TSEC 1–2 transmit error in) signals, 15-12
TSEC*n*_TXD[7:0] (TSEC 1–2 transmit data out) signals, 15-11

U

Universal serial bus, *see* USB interface
UPWAIT (LBC UPM wait) signal, 10-7, 10-60
USB interface
asynchronous schedule, 16-59, 16-93
adding queue heads, 16-94
empty detection, 16-97
list queue head pointer, 16-61
organization, 16-61
reclamation status bit, 16-98
removing queue heads, 16-95
traversal (start event), 16-98
block diagram, 16-2
dual-role controller
overview, 1-13
external port configurations, 16-3
features, 16-3
frame list
link pointer format, 16-60
functional descriptions
DMA engine, 16-58
FIFO RAM controller, 16-58
PHY interface, 16-58
system interface, 16-58
host data structures, 16-59
host operations, 16-79
behavior during wake-up events, 16-83
host controller initialization, 16-80
periodic schedule, 16-88
periodic schedule frame boundaries vs. bus frame boundaries, 16-85
port power control (PPC), 16-81
port suspend/resume, 16-82
reporting over-current, 16-81
schedule traversal rules, 16-84
split transactions, 16-103
execution state machine for isochronous, 16-123
isochronous, 16-118
periodic interrupt - Do-Complete-Split, 16-114
periodic interrupt - Do-Start-Split, 16-113
scheduling mechanisms for isochronous, 16-118
see also USB interface, split transactions
tests for execution, 16-114
tracking progress for isochronous transfers, 16-122
suspend/resume, 16-82
isochronous (high-speed) transfer descriptor (iTD), 16-59, 16-61
buffer page pointer list (plus), 16-63
host controller operational model, 16-89
managing transfers, 16-89
next link pointer, 16-62
periodic scheduling threshold, 16-92
software operational model, 16-91
transaction status and control list, 16-63
memory map/register definitions, 2-18–2-20, 16-16–16-18
modes of operation, 16-3
modules, 16-1
dual-role (DR), 16-1
multi-port host (MPH), 16-1
multi-port host controller
overview, 1-14
overview, 1-12, 16-2
periodic frame list, 16-59
periodic frame span traversal node (FSTN), 16-78

back path link pointer, 16-79
normal path pointer, 16-79
periodic schedule, 16-59
organization, 16-60
rebalancing, 16-117
queue element transfer descriptor (qTD), 16-59, 16-69
alternate next pointer, 16-70
buffer page pointer list, 16-73
next pointer, 16-69
token, 16-70
queue heads, 16-59, 16-74
endpoint capabilities/characteristics, 16-75
horizontal link pointer, 16-74
managing control/bulk/interrupt transfers, 16-98
buffer pointer list use for data streaming with qTDs, 16-99
ping control, 16-102
to the periodic schedule, 16-101
transfer complete interrupts, 16-101
managing the QH[FrameTag] field, 16-117
transfer overlay, 16-77
registers, 16-18–16-57
by acronym, *see* Register Index
capability registers, 16-18–16-23
operational registers, 16-23–16-57
signals, 16-4–16-15
PHY clocks, 16-15
serial interface, 16-11
serial OTG interface, 16-13
ULPI interface, 16-7
UTMI interface, 16-8
split transaction isochronous transfer descriptor (siTD), 16-59, 16-65
back link pointer, 16-68
buffer pointer list (plus), 16-68
endpoint capabilities/characteristics, 16-66
next link pointer, 16-65
transfer state, 16-67
split transactions, 16-103
asynchronous transfers, 16-103
Do-Complete-Split state, 16-104
Do-Start-Split state, 16-104
interrupt, 16-105
execution state machine, 16-112
host controller operational model for FSTNs, 16-108
scheduling mechanisms, 16-106
software operational model for FSTNs, 16-110
tracking progress for interrupt transfers, 16-111
User-level SPRs, 7-15

W

Watchdog timer (WDT), 5-31
block diagram, 5-32
features, 5-32
functional block diagram, 5-36
functional description, 5-35
memory map/register definition, 5-33
modes of operation, 5-32, 5-37
overview, 5-31
registers, 5-33–5-35

Z

ZBT SRAM interface (LBC), 10-94

Preface PRE
Overview 1
Memory Map 2
Signal Descriptions 3
Reset, Clocking, and Initialization 4
System Configuration 5
Arbiter and Bus Monitor 6
e300 Processor Core 7
Integrated Programmable Interrupt Controller (IPIC) 8
DDR Memory Controller 9
Local Bus Controller 10
Sequencer 11
DMA 12
PCI Bus Interface 13
Security Engine (SEC) 2.0 14
Three-Speed Ethernet Controllers 15
Universal Serial Bus Interface 16
I^2C Interfaces 17
DUART 18
Serial Peripheral Interface 19
JTAG/Testing Support 20
General Purpose I/O (GPIO) 21
Delay Lock Loop (DLL) 22

Revision History APP
Glossary GLO
Register Index REG
General Index IND

PRE Preface

1 Overview

2 Memory Map

3 Signal Descriptions

4 Reset, Clocking, and Initialization

5 System Configuration

6 Arbiter and Bus Monitor

7 e300 Processor Core

8 Integrated Programmable Interrupt Controller (IPIC)

9 DDR Memory Controller

10 Local Bus Controller

11 Sequencer

12 DMA

13 PCI Bus Interface

14 Security Engine (SEC) 2.0

15 Three-Speed Ethernet Controller

16 Universal Serial Bus Interface

17 I^2C Interface

18 DUART

19 Serial Peripheral Interface

20 JTAG/Testing Support

21 General Purpose I/O (GPIO)

22 Delay Lock Loop (DLL)

APP Revision History

GLO Glossary

REG Register Index

IND General Index