

MPC8309 PowerQUICC II Pro Integrated Communications Processor Reference Manual

MPC8309RM
Rev. 2
10/2014

How to Reach Us:

Home Page:

www.freescale.com

Web Support:

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

USA/Europe or Locations Not Listed:

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

Europe, Middle East, and Africa:

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

Japan:

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

Asia/Pacific:

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

For Literature Requests Only:

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

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

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

Freescale, the Freescale logo, CodeWarrior, ColdFire, PowerQUICC, StarCore, and Symphony are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. CoreNet, QorIQ, QUICC Engine, and VortiQa are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.
© 2012-2014 Freescale Semiconductor, Inc.



Contents

Paragraph Number	Title	Page Number
About This Book		
	Organization.....	xlix
	Suggested Reading.....	li
	General Information.....	li
	Related Documentation.....	li
	Conventions	lii
	Signal Conventions	liii
	Acronyms and Abbreviations	liii

Chapter 1 Overview

1.1	MPC8309 PowerQUICC II Pro Processor Overview.....	1-1
1.2	Features	1-2
1.3	MPC8309 Architecture Overview	1-7
1.3.1	Power Architecture Core	1-7
1.3.2	QUICC Engine Block.....	1-10
1.3.3	DDR2 Memory Controller.....	1-14
1.3.4	PCI Bus Interface.....	1-14
1.3.5	I/O Sequencer	1-15
1.3.6	Enhanced Local Bus Controller (eLBC).....	1-15
1.3.7	Integrated Programmable Interrupt Controller (IPIC).....	1-17
1.3.8	Enhanced Secure Digital Host Controller (eSDHC).....	1-17
1.3.9	Universal Serial Bus (USB) 2.0.....	1-18
1.3.10	FlexCAN Module	1-19
1.3.11	Dual I ² C Interfaces	1-20
1.3.12	DMA Engine 1.....	1-20
1.3.13	DMA Engine 2.....	1-21
1.3.14	Dual Universal Asynchronous Receiver/Transmitter (DUART).....	1-21
1.3.15	Serial Peripheral Interface (SPI).....	1-22
1.3.16	System Timers	1-22
1.3.17	Real Time Clock	1-22

Chapter 2 Memory Map

2.1	Internal Memory Mapped Registers	2-1
-----	--	-----

Contents

Paragraph Number	Title	Page Number
2.2	Accessing IMMR Memory from the Local Processor	2-1
2.3	IMMR Address Map	2-1

Chapter 3 Signal Descriptions

3.1	Signals Overview	3-1
3.2	Output Signal States During Reset	3-20

Chapter 4 Reset, Clocking, and Initialization

4.1	External Signals	4-1
4.1.1	Reset Signals	4-1
4.1.2	Clock Signals	4-2
4.2	Functional Description	4-4
4.2.1	Reset Operations	4-4
4.2.2	Power-On Reset Flow	4-6
4.2.3	Hard Reset Flow	4-7
4.3	Reset Configuration	4-8
4.3.1	Reset Configuration Signals	4-8
4.3.2	Reset Configuration Words	4-10
4.3.3	Loading the Reset Configuration Words	4-17
4.4	Clocking	4-25
4.4.1	System Clock Domains	4-26
4.5	Memory Map/Register Definitions	4-27
4.5.1	Reset Configuration Register Descriptions	4-27
4.5.2	Clock Configuration Registers	4-31

Chapter 5 System Boot

5.1	Bootting from On Chip ROM	5-1
5.2	eSDHC Boot	5-1
5.2.1	Overview	5-2
5.2.2	Features	5-2
5.2.3	SD/MMC Card Data Structure	5-3
5.2.4	eSDHC Controller Initial Configuration	5-7
5.2.5	eSDHC Controller Boot Sequence	5-7
5.2.6	eSDHC Boot Error Handling	5-8
5.3	SPI Boot ROM	5-9

Contents

Paragraph Number	Title	Page Number
5.3.1	Overview.....	5-10
5.3.2	Features.....	5-10
5.3.3	EEPROM Data Structure	5-10
5.3.4	SPI Controller Configuration.....	5-13

Chapter 6 System Configuration

6.1	Introduction.....	6-1
6.2	Local Memory Map Overview and Example	6-1
6.2.1	Address Translation and Mapping	6-3
6.2.2	Window into Configuration Space.....	6-4
6.2.3	Local Access Windows.....	6-4
6.2.4	Local Access Register Descriptions	6-6
6.2.5	Precedence of Local Access Windows	6-14
6.2.6	Configuring Local Access Windows	6-14
6.2.7	Distinguishing Local Access Windows from Other Mapping Functions	6-14
6.2.8	hasOutbound Address Translation and Mapping Windows	6-15
6.2.9	Inbound Address Translation and Mapping Windows	6-15
6.2.10	hasInternal Memory Map.....	6-15
6.2.11	Accessing Internal Memory from External Masters.....	6-16
6.3	System Configuration	6-16
6.3.1	System Configuration Register Memory Map.....	6-16
6.3.2	System Configuration Registers	6-17
6.3.3	Multisite Muxing	6-39
6.4	Software Watchdog Timer (WDT).....	6-40
6.4.1	WDT Overview.....	6-40
6.4.2	WDT Features.....	6-41
6.4.3	WDT Modes of Operation	6-41
6.4.4	WDT Memory Map/Register Definition	6-41
6.4.5	Functional Description.....	6-44
6.4.6	Initialization/Application Information (WDT Programming Guidelines).....	6-46
6.5	Real Time Clock Module (RTC).....	6-47
6.5.1	Overview.....	6-47
6.5.2	Features.....	6-47
6.5.3	Modes of Operation	6-48
6.5.4	External Signal Description	6-48
6.5.5	RTC Memory Map/Register Definition.....	6-48
6.5.6	Functional Description.....	6-52
6.5.7	RTC Initialization Sequence	6-53
6.6	Periodic Interval Timer (PIT)	6-54

Contents

Paragraph Number	Title	Page Number
6.6.1	PIT Overview.....	6-54
6.6.2	PIT Features.....	6-54
6.6.3	PIT Modes of Operation.....	6-54
6.6.4	PIT External Signal Description.....	6-55
6.6.5	PIT Memory Map/Register Definition.....	6-55
6.6.6	Functional Description.....	6-58
6.6.7	PIT Programming Guidelines.....	6-59
6.7	General-Purpose Timers (GTMs).....	6-60
6.7.1	GTM Overview.....	6-60
6.7.2	GTM Features.....	6-60
6.7.3	GTM Modes of Operation.....	6-61
6.7.4	GTM External Signal Description.....	6-62
6.7.5	GTM Memory Map/Register Definition.....	6-64
6.7.6	Functional Description.....	6-73
6.7.7	Initialization/Application Information (Programming Guidelines for GTM Registers) ... 6-76	6-76
6.8	Power Management Control (PMC).....	6-76
6.8.1	External Signal Description.....	6-77
6.8.2	PMC Memory Map/Register Definition.....	6-77
6.8.3	Functional Description.....	6-78

Chapter 7 Arbiter and Bus Monitor

7.1	Overview.....	7-1
7.1.1	Coherent System Bus Overview.....	7-1
7.2	Arbiter Memory Map/Register Definition.....	7-2
7.2.1	Arbiter Configuration Register (ACR).....	7-3
7.2.2	Arbiter Timers Register (ATR).....	7-4
7.2.3	Arbiter Event Register (AER).....	7-5
7.2.4	Arbiter Interrupt Definition Register (AIDR).....	7-6
7.2.5	Arbiter Mask Register (AMR).....	7-7
7.2.6	Arbiter Event Attributes Register (AEATR).....	7-8
7.2.7	Arbiter Event Address Register (AEADR).....	7-9
7.2.8	Arbiter Event Response Register (AERR).....	7-10
7.3	Functional Description.....	7-11
7.3.1	Arbitration Policy.....	7-11
7.3.2	Bus Error Detection.....	7-14
7.4	Initialization/Applications Information.....	7-17
7.4.1	Initialization Sequence.....	7-17
7.4.2	Error Handling Sequence.....	7-17

Contents

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

Chapter 8 e300 Processor Core Overview

8.1	Overview	8-1
8.1.1	Features	8-3
8.1.2	Instruction Unit	8-6
8.1.3	Independent Execution Units	8-7
8.1.4	Completion Unit	8-8
8.1.5	Memory Subsystem Support	8-8
8.1.6	Bus Interface Unit (BIU)	8-10
8.1.7	System Support Functions	8-11
8.2	e300 Processor and System Version Numbers	8-13
8.3	PowerPC Architecture Implementation	8-13
8.4	Implementation-Specific Information	8-14
8.4.1	Register Model	8-14
8.4.2	Instruction Set and Addressing Modes	8-26
8.4.3	Cache Implementation	8-29
8.4.4	Interrupt Model	8-31
8.4.5	Memory Management	8-35
8.4.6	Instruction Timing	8-36
8.4.7	Core Interface	8-37
8.4.8	Debug Features	8-39
8.5	Differences Between Cores	8-40

Chapter 9 Integrated Programmable Interrupt Controller (IPIC)

9.1	Introduction	9-1
9.2	Features	9-4
9.3	Modes of Operation	9-4
9.3.1	Core Enable Mode	9-4
9.3.2	Core Disable Mode	9-4
9.4	External Signal Description	9-5
9.4.1	Overview	9-5
9.4.2	Detailed Signal Descriptions	9-5
9.5	Memory Map/Register Definition	9-5
9.5.1	System Global Interrupt Configuration Register (SICFR)	9-7
9.5.2	System Global Interrupt Vector Register (SIVCR)	9-8
9.5.3	System Internal Interrupt Pending Registers (SIPNR_H and SIPNR_L)	9-11
9.5.4	System Internal Interrupt Group A Priority Register (SIPRR_A)	9-14
9.5.5	System Internal Interrupt Group B Priority Register (SIPRR_B)	9-15

Contents

Paragraph Number	Title	Page Number
9.5.6	System Internal Interrupt Group C Priority Register (SIPRR_C)	9-15
9.5.7	System Internal Interrupt Group D Priority Register (SIPRR_D)	9-16
9.5.8	System Internal Interrupt Mask Register (SIMSR_H and SIMSR_L)	9-17
9.5.9	System Internal Interrupt Control Register (SICNR)	9-18
9.5.10	System External Interrupt Pending Register (SEPNR).....	9-20
9.5.11	System Mixed Interrupt Group A Priority Register (SMPRR_A).....	9-21
9.5.12	System Mixed Interrupt Group B Priority Register (SMPRR_B)	9-22
9.5.13	System External Interrupt Mask Register (SEMSR)	9-22
9.5.14	System External Interrupt Control Register (SECNR).....	9-23
9.5.15	System Error Status Register (SERSR)	9-25
9.5.16	System Error Mask Register (SERMR).....	9-26
9.5.17	System Error Control Register (SERCR)	9-27
9.5.18	System External interrupt Polarity Control Register (SEPCR)	9-27
9.5.19	System Internal Interrupt Force Registers (SIFCR_H and SIFCR_L)	9-28
9.5.20	System External Interrupt Force Register (SEFCR).....	9-30
9.5.21	System Error Force Register (SERFR).....	9-30
9.5.22	System Critical Interrupt Vector Register (SCVCR)	9-31
9.5.23	System Management Interrupt Vector Register (SMVCR)	9-31
9.5.24	QUICC Engine Ports Interrupt Event Register (CEPIER)	9-32
9.5.25	QUICC Engine Ports Interrupt Mask Register (CEPIMR).....	9-33
9.5.26	QUICC Engine Ports Interrupt Control Register (CEPICR)	9-35
9.6	Functional Description.....	9-35
9.6.1	Interrupt Types	9-35
9.6.2	Interrupt Configuration	9-36
9.6.3	Internal Interrupts Group Relative Priority.....	9-37
9.6.4	Mixed Interrupts Group Relative Priority.....	9-37
9.6.5	Highest Priority Interrupt.....	9-38
9.6.6	Interrupt Source Priorities.....	9-38
9.6.7	Masking Interrupt Sources.....	9-42
9.6.8	Interrupt Vector Generation and Calculation	9-42
9.6.9	Machine Check Interrupts.....	9-43

Chapter 10 DDR Memory Controller

10.1	Introduction.....	10-1
10.2	Features	10-2
10.2.1	Modes of Operation	10-3
10.3	External Signal Descriptions	10-3
10.3.1	Signals Overview	10-3
10.3.2	Detailed Signal Descriptions	10-6

Contents

Paragraph Number	Title	Page Number
10.4	Memory Map/Register Definition	10-9
10.4.1	Register Descriptions	10-11
10.5	Functional Description.....	10-38
10.5.1	DDR SDRAM Interface Operation.....	10-42
10.5.2	DDR SDRAM Address Multiplexing.....	10-43
10.5.3	JEDEC Standard DDR SDRAM Interface Commands	10-45
10.5.4	DDR SDRAM Interface Timing	10-47
10.5.5	DDR SDRAM Mode-Set Command Timing.....	10-51
10.5.6	DDR SDRAM Registered DIMM Mode	10-51
10.5.7	DDR SDRAM Write Timing Adjustments	10-52
10.5.8	DDR SDRAM Refresh	10-53
10.5.9	DDR Data Beat Ordering.....	10-56
10.5.10	Page Mode and Logical Bank Retention	10-57
10.5.11	Error Checking and Correcting (ECC)	10-58
10.5.12	Error Management	10-60
10.6	Initialization/Application Information	10-60
10.6.1	DDR SDRAM Initialization Sequence	10-62

Chapter 11 Enhanced Local Bus Controller

11.1	Introduction.....	11-1
11.1.1	Overview.....	11-2
11.1.2	Features	11-2
11.1.3	Modes of Operation	11-3
11.2	External Signal Descriptions	11-4
11.3	Memory Map/Register Definition	11-8
11.3.1	Register Descriptions	11-9
11.4	Functional Description.....	11-38
11.4.1	Basic Architecture.....	11-40
11.4.2	General-Purpose Chip-Select Machine (GPCM).....	11-43
11.4.3	Flash Control Machine (FCM)	11-55
11.4.4	User-Programmable Machines (UPMs).....	11-71
11.5	Initialization/Application Information	11-87
11.5.1	Interfacing to Peripherals in Different Address Modes	11-87
11.5.2	Bus Turnaround	11-89
11.5.3	Interface to Different Port-Size Devices.....	11-90
11.5.4	Command Sequence Examples for NAND Flash EEPROM.....	11-91
11.5.5	Interfacing to Fast-Page Mode DRAM Using UPM	11-95
11.5.6	Interfacing to ZBT SRAM Using UPM.....	11-100

Contents

Paragraph Number	Title	Page Number
Chapter 12		
Enhanced Secure Digital Host Controller		
12.1	Overview.....	12-1
12.2	Features.....	12-3
12.2.1	Data Transfer Modes.....	12-4
12.3	External Signal Description.....	12-4
12.4	Memory Map/Register Definition.....	12-5
12.4.1	DMA System Address Register (DSADDR).....	12-7
12.4.2	Block Attributes Register (BLKATTR).....	12-7
12.4.3	Command Argument Register (CMDARG).....	12-8
12.4.4	Transfer Type Register (XFERTYP).....	12-9
12.4.5	Command Response 0–3 (CMDRSP0–3).....	12-12
12.4.6	Buffer Data Port Register (DATPORT).....	12-14
12.4.7	Present State Register (PRSSTAT).....	12-15
12.4.8	Protocol Control Register (PROCTL).....	12-19
12.4.9	System Control Register (SYSCTL).....	12-22
12.4.10	Interrupt Status Register (IRQSTAT).....	12-24
12.4.11	Interrupt Status Enable Register (IRQSTATEN).....	12-29
12.4.12	Interrupt Signal Enable Register (IRQSIGEN).....	12-31
12.4.13	Auto CMD12 Error Status Register (AUTOC12ERR).....	12-33
12.4.14	Host Controller Capabilities (HOSTCAPBLT).....	12-35
12.4.15	Watermark Level Register (WML).....	12-36
12.4.16	Force Event Register (FEVT).....	12-37
12.4.17	Host Controller Version Register (HOSTVER).....	12-39
12.4.18	DMA Control Register (DCR).....	12-39
12.5	Functional Description.....	12-39
12.5.1	Data Buffer.....	12-39
12.5.2	DMA CSB Interface.....	12-42
12.5.3	SD Protocol Unit.....	12-43
12.5.4	Clock & Reset Manager.....	12-45
12.5.5	Clock Generator.....	12-45
12.5.6	SDIO Card Interrupt.....	12-45
12.5.7	Card Insertion and Removal Detection.....	12-47
12.5.8	Power Management.....	12-47
12.6	Initialization/Application Information.....	12-48
12.6.1	Command Send and Response Receive Basic Operation.....	12-48
12.6.2	Card Identification Mode.....	12-49
12.6.3	Card Access.....	12-53
12.6.4	Switch Function.....	12-58
12.6.5	Commands for MMC/SD/SDIO.....	12-60

Contents

Paragraph Number	Title	Page Number
12.7	Software Restrictions	12-66
12.7.1	Initialization Active	12-66
12.7.2	Software Polling Procedure	12-66
12.7.3	Suspend Operation	12-66
12.7.4	Data Port Access	12-66
12.7.5	Multi-block Read	12-66

Chapter 13 DMA Engine 1

13.1	Overview	13-2
13.1.1	Features	13-2
13.2	DMAC Memory Map/Register Definition	13-2
13.2.1	DMA Control Register (DMACR)	13-4
13.3	DMA Error Status (DMAES)	13-5
13.3.1	DMA Enable Request Register (DMAERQ)	13-7
13.3.2	DMA Enable Error Interrupt Register (DMAEEI)	13-8
13.3.3	DMA Set Enable Error Interrupt (DMASEEI)	13-9
13.3.4	DMA Clear Enable Error Interrupt (DMACEEI)	13-10
13.3.5	DMA Clear Interrupt Request (DMACINT)	13-10
13.3.6	DMA Clear Error (DMACERR)	13-11
13.3.7	DMA Set START Bit (DMASSRT)	13-12
13.3.8	DMA Clear DONE Status (DMACDNE)	13-12
13.3.9	DMA Interrupt Request Register (DMAINT)	13-13
13.3.10	DMA Error Register (DMAERR)	13-14
13.3.11	DMA General Purpose Output Register (DMAGPOR)	13-15
13.3.12	DMA Channel <i>n</i> Priority (DCHPRI _{<i>n</i>}), <i>n</i> = 0–15	13-16
13.3.13	Transfer Control Descriptor (TCD)	13-17
13.4	Functional Description	13-25
13.4.1	DMA Microarchitecture	13-25
13.4.2	DMA Basic Data Flow	13-26
13.5	Initialization/Application Information	13-29
13.5.1	DMA Initialization	13-29
13.5.2	DMA Programming Errors	13-30
13.6	DMA Transfer	13-30
13.6.1	Single Request	13-30
13.6.2	Multiple Requests	13-31
13.7	TCD Status	13-33
13.7.1	Minor Loop Complete	13-33
13.7.2	Active Channel TCD Reads	13-33
13.7.3	Preemption status	13-33

Contents

Paragraph Number	Title	Page Number
13.8	Channel Linking	13-34
13.9	Programming during channel execution	13-34
13.9.1	Dynamic priority changing	13-34
13.9.2	Dynamic channel linking and dynamic scatter/gather	13-35

Chapter 14 DMA Engine 2

14.1	DMA Features	14-1
14.2	DMA Memory Map/Register Definition	14-2
14.3	DMA Register Descriptions	14-3
14.3.1	Outbound Message Interrupt Status Register (OMISR)	14-3
14.3.2	Outbound Message Interrupt Mask Register (OMIMR)	14-4
14.3.3	Inbound Message Registers (IMR0–IMR1)	14-5
14.3.4	Outbound Message Registers (OMR0–OMR1)	14-5
14.3.5	Doorbell Registers	14-6
14.3.6	Inbound Message Interrupt Status Register (IMISR)	14-7
14.3.7	Inbound Message Interrupt Mask Register (IMIMR)	14-8
14.3.8	DMA Registers	14-9
14.4	Functional Description	14-15
14.4.1	Message Unit	14-15
14.4.2	DMA Controller	14-16
14.4.3	DMA Operation	14-16
14.4.4	DMA Segment Descriptors	14-18
14.5	Initialization/Application Information	14-20
14.5.1	Initialization Steps in Direct Mode	14-20
14.5.2	Initialization Steps in Chaining Mode	14-20

Chapter 15 FlexCAN

15.1	Introduction	15-1
15.1.1	Overview	15-2
15.1.2	FlexCAN Module Features	15-3
15.1.3	Modes of Operation	15-4
15.2	External Signal Description	15-5
15.2.1	Signals Overview	15-5
15.3	Memory Map/Register Definition	15-5
15.3.1	Message Buffer Structure	15-7
15.3.2	Rx FIFO Structure	15-10
15.3.3	Register Descriptions	15-12

Contents

Paragraph Number	Title	Page Number
15.4	Functional Description.....	15-28
15.4.1	Overview.....	15-28
15.4.2	Transmit Process.....	15-29
15.4.3	Arbitration process.....	15-30
15.4.4	Receive Process	15-30
15.4.5	Matching Process.....	15-32
15.4.6	Data Coherence.....	15-33
15.4.7	Rx FIFO	15-36
15.4.8	CAN Protocol Related Features.....	15-36
15.4.9	Modes of Operation Details.....	15-40
15.4.10	Interrupts.....	15-41
15.4.11	Bus Interface	15-42
15.5	Initialization/Application Information.....	15-42
15.5.1	FlexCAN Initialization Sequence	15-42
15.5.2	FlexCAN Addressing and RAM size configurations	15-44

Chapter 16 Universal Serial Bus Interface

16.1	Introduction.....	16-1
16.1.1	Overview.....	16-2
16.1.2	Features.....	16-2
16.1.3	Modes of Operation	16-2
16.2	External Signals	16-2
16.2.1	ULPI Interface	16-3
16.3	Memory Map/Register Definitions	16-4
16.3.1	Capability Registers.....	16-6
16.3.2	Operational Registers.....	16-10
16.4	Functional Description.....	16-44
16.4.1	System Interface	16-44
16.4.2	DMA Engine.....	16-44
16.4.3	FIFO RAM Controller	16-45
16.4.4	PHY Interface	16-45
16.5	Host Data Structures	16-45
16.5.1	Periodic Frame List.....	16-46
16.5.2	Asynchronous List Queue Head Pointer.....	16-47
16.5.3	Isochronous (High-Speed) Transfer Descriptor (iTd).....	16-47
16.5.4	Split Transaction Isochronous Transfer Descriptor (siTD).....	16-51
16.5.5	Queue Element Transfer Descriptor (qTD)	16-55
16.5.6	Queue Head.....	16-60
16.5.7	Periodic Frame Span Traversal Node (FSTN).....	16-64

Contents

Paragraph Number	Title	Page Number
16.6	Host Operations	16-65
16.6.1	Host Controller Initialization	16-66
16.6.2	Power Port.....	16-67
16.6.3	Reporting Over-Current	16-67
16.6.4	Suspend/Resume	16-67
16.6.5	Schedule Traversal Rules.....	16-69
16.6.6	Periodic Schedule Frame Boundaries vs. Bus Frame Boundaries.....	16-71
16.6.7	Periodic Schedule	16-73
16.6.8	Managing Isochronous Transfers Using iTDs	16-74
16.6.9	Asynchronous Schedule.....	16-79
16.6.10	Managing Control/Bulk/Interrupt Transfers via Queue Heads.....	16-83
16.6.11	Ping Control.....	16-87
16.6.12	Split Transactions.....	16-88
16.6.13	Port Test Modes	16-116
16.6.14	Interrupts	16-117
16.7	Device Data Structures	16-121
16.7.1	Endpoint Queue Head.....	16-122
16.7.2	Endpoint Transfer Descriptor (dTd)	16-124
16.8	Device Operational Model.....	16-127
16.8.1	Device Controller Initialization	16-127
16.8.2	Port State and Control.....	16-128
16.8.3	Managing Endpoints	16-131
16.8.4	Managing Queue Heads.....	16-141
16.8.5	Managing Transfers with Transfer Descriptors	16-143
16.8.6	Servicing Interrupts.....	16-146
16.9	Deviations from the EHCI Specifications	16-147
16.9.1	Embedded Transaction Translator Function.....	16-148
16.9.2	Device Operation	16-151
16.9.3	Non-Zero Fields the Register File	16-152
16.9.4	SOF Interrupt	16-152
16.9.5	Embedded Design.....	16-152
16.9.6	Miscellaneous Variations from EHCI.....	16-152

Chapter 17 I²C 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

Contents

Paragraph Number	Title	Page Number
17.2.2	Detailed Signal Descriptions	17-3
17.3	Memory Map/Register Definition	17-4
17.3.1	Register Descriptions	17-5
17.4	Functional Description.....	17-10
17.4.1	Transaction Protocol	17-10
17.4.2	Arbitration Procedure	17-14
17.4.3	Handshaking	17-15
17.4.4	Clock Control.....	17-15
17.4.5	Boot Sequencer Mode.....	17-16
17.5	Initialization/Application Information	17-20
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

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
18.4	Functional Description.....	18-18
18.4.1	Serial Interface.....	18-19
18.4.2	Baud-Rate Generator Logic	18-20
18.4.3	Local Loopback Mode	18-21
18.4.4	Errors	18-21
18.4.5	FIFO Mode	18-21
18.5	DUART Initialization/Application Information	18-23

Chapter 19 Serial Peripheral Interface

Contents

Paragraph Number	Title	Page Number
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
19.3	External Signal Descriptions.....	19-6
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
19.5	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
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/DIR–GP2DIR).....	21-3
21.3.2	GPIO _n Open Drain Register (GP/ODR–GP2ODR).....	21-4
21.3.3	GPIO _n Data Register (GP/DAT–GP2DAT).....	21-4
21.3.4	GPIO _n Interrupt Event Register (GP/IER–GP2IER).....	21-5
21.3.5	GPIO _n Interrupt Mask Register (GP/IMR–GP2IMR).....	21-5
21.3.6	GPIO _n Interrupt Control Register (GP/ICR–GP2ICR).....	21-6

Chapter 22 Sequencer

Contents

Paragraph Number	Title	Page Number
22.1	Sequencer Overview	22-1
22.1.1	Sequencer Features	22-2
22.2	Sequencer External Signal Description	22-2
22.3	Sequencer Memory Map/Register Definition	22-2
22.4	Sequencer Register Descriptions	22-3
22.4.1	PCI Outbound Translation Address Registers (POTAR _n).....	22-3
22.4.2	PCI Outbound Base Address Registers (POBAR _n)	22-3
22.4.3	PCI Outbound Comparison Mask Registers (POCMR _n)	22-4
22.4.4	Power Management Control Register (PMCR)	22-5
22.4.5	Discard Timer Control Register (DTCR)	22-6
22.5	Functional Description.....	22-6
22.5.1	Transaction Forwarding	22-6
22.5.2	PCI Outbound Address Translation	22-7
22.5.3	Transaction Ordering	22-8

Chapter 23 PCI Bus Interface

23.1	PCI Introduction	23-1
23.1.1	PCI Features.....	23-3
23.1.2	PCI Modes of Operation	23-3
23.2	PCI External Signal Description.....	23-4
23.3	PCI Memory Map/Register Definitions.....	23-11
23.3.1	PCI Configuration Access Registers.....	23-12
23.3.2	PCI Memory-Mapped Control and Status BE Registers	23-15
23.3.3	PCI Configuration Space Registers	23-25
23.4	Functional Description.....	23-40
23.4.1	PCI Bus Arbitration	23-40
23.4.2	Bus Commands	23-43
23.4.3	PCI Protocol Fundamentals	23-44
23.4.4	Other Bus Operations.....	23-50
23.4.5	Error Functions	23-54
23.4.6	PCI Inbound Address Translation.....	23-56
23.4.7	CompactPCI Hot Swap Specification Support	23-57
23.4.8	Byte Ordering	23-57
23.5	Initialization/Application Information	23-59
23.5.1	Initialization Sequence for Host Mode	23-59
23.5.2	Initialization Sequence for Agent Mode	23-59

Chapter 24 QUICC Engine Block on the MPC8309

Contents

Paragraph Number	Title	Page Number
24.1	QUICC Engine Block	24-1
24.2	QUICC Engine Implementation Details for the MPC8309	24-2
24.2.1	System Interface	24-4
24.2.2	Configuration – Parameter RAM.....	24-35
24.2.3	QUICC Engine Multiplexing and Timers.....	24-40
24.2.4	UCC Ethernet (UEC).....	24-43
24.2.5	IEEE Standard 1588 Assist.....	24-43

Appendix A

Complete List of Configuration, Control, and Status Registers

A.1	Local Access Windows	1-1
A.2	System Configuration Registers	1-2
A.3	Watchdog Timer (WDT)	1-3
A.4	Real Time Clock (RTC)	1-4
A.5	Periodic Interval Timer (PIT)	1-4
A.6	General Purpose (Global) Timers (GTMs)	1-4
A.7	Integrated Programmable Interrupt Controller (IPIC)	1-5
A.8	QUICC Engine Ports Interrupts	1-7
A.9	System Arbiter	1-7
A.10	Reset Configuration	1-7
A.11	Clock Configuration	1-8
A.12	Power Management Controller (PMC).....	1-8
A.13	General Purpose I/O (GPIO).....	1-8
A.14	DDR Memory Controller.....	1-9
A.15	I ² C Controller	1-11
A.16	DUART	1-11
A.17	Enhanced Local Bus Controller (eLBC).....	1-12
A.18	Serial Peripheral Interface (SPI)	1-14
A.19	DMA Engine 1	1-14
A.20	DMA Engine 2.....	1-15
A.21	Enhanced Secure Digital Host Controller (eSDHC).....	1-16
A.22	FlexCAN	1-17
A.23	PCI Configuration Access Registers.....	1-18
A.24	PCI Memory Mapped Registers	1-19
A.25	Universal Serial Bus (USB) Interface.....	1-20

Appendix B

Revision History

B.1	Changes From Revision 1 to Revision 2	2-1
-----	---	-----

Contents

Paragraph Number	Title	Page Number
B.2	Changes From Revision 0 to Revision 1	2-7

Contents

**Paragraph
Number**

Title

**Page
Number**

Figures

Figure Number	Title	Page Number
1-1	MPC8309 Block Diagram.....	1-2
1-2	MPC8309 Integrated e300 Core Block Diagram.....	1-9
1-3	QUICC Engine Block Architectural Block Diagram.....	1-11
1-4	USB Controllers Port Configuration.....	1-19
3-1	MPC8309 Signal Groupings (1 of 3).....	3-2
3-2	MPC8309 Signal Groupings (2 of 3).....	3-3
3-3	MPC8309 Signal Groupings (3 of 3).....	3-4
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	EEPROM Data Format for Reset Configuration Words Preload Command	4-21
4-6	EEPROM Contents	4-22
4-7	MPC8309 Clock Subsystem	4-25
4-8	Reset Status Register (RSR).....	4-28
4-9	Reset Mode Register (RMR).....	4-29
4-10	Reset Protection Register (RPR).....	4-30
4-11	Reset Control Register (RCR).....	4-30
4-12	Reset Control Enable Register (RCER).....	4-31
4-13	System PLL Mode Register	4-32
4-14	Output Clock Control Register (OCCR).....	4-33
4-15	System Clock Control Register (SCCR).....	4-34
5-1	SD/MMC Card Data Structure.....	5-3
5-2	Config Address Fields.....	5-5
5-3	SD/MMC Card Data Structure for Maximum Redundancy	5-9
5-4	SPI EEPROM Data Structure.....	5-11
5-5	Config Address Fields.....	5-12
5-6	External Signal Connection	5-14
6-1	Local Memory Map Example	6-2
6-2	Internal Memory Map Registers' Base Address Register (IMMRBAR).....	6-7
6-3	Alternate Configuration Base Address Register (ALTCBAR)	6-7
6-4	LBC Local Access Window <i>n</i> Base Address Registers (LBLAWBAR0–LBLAWBAR3)	6-8
6-5	LBC Local Access Window <i>n</i> Attributes Registers (LBLAWAR0–LBLAWAR3)	6-9
6-6	PCI Local Access Window <i>n</i> Base Address Registers (PCILAWBAR0–PCILAWBAR1) .	6-10
6-7	PCI Local Access Window <i>n</i> Attributes Registers (PCILAWAR0–PCILAWAR1).....	6-11
6-8	DDR Local Access Window <i>n</i> Base Address Registers (DDRLAWBAR0–DDRLAWBAR1)...	6-12
6-9	DDR Local Access Window <i>n</i> Attributes Registers (DDRLAWAR0–DDRLAWAR1).....	6-13
6-10	System General Purpose Register Low (SGPRL).....	6-17
6-11	System General Purpose Register High (SGPRH)	6-18
6-12	System Part and Revision ID Register (SPRIDR)	6-18

Figures

Figure Number	Title	Page Number
6-13	System Priority Configuration Register (SPCR)	6-20
6-14	System I/O Configuration Register 1 (SICR_1)	6-21
6-15	System I/O Configuration Register 2 (SICR_2)	6-25
6-16	DDR Control Driver Register (DDRCDDR)	6-29
6-17	DDR Debug Status Register (DDRDSR).....	6-30
6-18	eSDHC Control Register (SDHCCR)	6-31
6-19	CAN Access Control Register (CAN_DBG_CTRL).....	6-32
6-20	SPI Chip Select Register (SPI_CS).....	6-33
6-21	General Purpose Register 1 (GPR_1)	6-34
6-22	CAN Interrupt Status Register (CAN_INT_STAT)	6-36
6-23	DUART Interrupt Status Register (DUART_INT_STAT)	6-38
6-24	GPIO Interrupt Status Register (GPIO_INT_STAT)	6-39
6-25	Software Watchdog Timer High-Level Block Diagram	6-40
6-26	System Watchdog Control Register (SWCRR).....	6-42
6-27	System Watchdog Count Register (SWCNR).....	6-43
6-28	System Watchdog Service Register (SWSRR)	6-43
6-29	Software Watchdog Timer Service State Diagram.....	6-45
6-30	Software Watchdog Timer Functional Block Diagram.....	6-45
6-31	Real Time Clock Module High Level Block Diagram	6-47
6-32	Real Time Counter Control Register (RTCNR).....	6-49
6-33	Real Time Counter Load Register (RTLDR)	6-49
6-34	Real Time Counter Prescale Register (RTPSR).....	6-50
6-35	Real Time Counter Register (RTCTR).....	6-50
6-36	Real Time Counter Event Register (RTEVR).....	6-51
6-37	Real Time Counter Alarm Register (RTALR)	6-51
6-38	Real Time Clock Module Functional Block Diagram	6-52
6-39	Periodic Interval Timer High Level Block Diagram.....	6-54
6-40	Periodic Interval Timer Control Register (PTCNR)	6-56
6-41	Periodic Interval Timer Load Register (PTLDR)	6-56
6-42	Periodic Interval Timer Prescale Register (PTPSR)	6-57
6-43	Periodic Interval Timer Counter Register (PTCTR).....	6-57
6-44	Periodic Interval Timer Event Register (PTEVR)	6-58
6-45	Periodic Interval Timer Functional Block Diagram.....	6-59
6-46	Global Timers Block Diagram	6-60
6-47	Global Timers Configuration Register 1 (GTCFR1)	6-66
6-48	Global Timers Configuration Register 2 (GTCFR2)	6-67
6-49	Global Timers Mode Registers (GTMDR1–GTMDR4).....	6-69
6-50	Global Timers Reference Registers (GTRFR1–GTRFR4)	6-70
6-51	Global Timers Capture Registers (GTCPR1–GTCPR4).....	6-70
6-52	Global Timers Counter Registers (GTCNR1–GTCNR4).....	6-71
6-53	Global Timers Event Registers (GTEVR1–GTEVR4)	6-71

Figures

Figure Number	Title	Page Number
6-54	Global Timers Prescale Registers (GTPSR1–GTPSR4).....	6-72
6-55	Timers Non-Cascaded Mode Block Diagram	6-74
6-56	Timer Pair-Cascaded Mode Block Diagram	6-75
6-57	Timers Super-Cascaded Mode Block Diagram.....	6-75
6-58	Power Management Controller Configuration Register	6-77
7-1	Arbiter Configuration Register (ACR)	7-3
7-2	Arbiter Timers Register (ATR)	7-4
7-3	Arbiter Event Register (AER).....	7-5
7-4	Arbiter Interrupt Definition Register (AIDR).....	7-6
7-5	Arbiter Mask Register (AMR)	7-7
7-6	Arbiter Event Attributes Register (AEATR).....	7-8
7-7	Arbiter Event Address Register (AEADR).....	7-10
7-8	Arbiter Event Response Register (AERR).....	7-10
7-9	Address Bus Arbitration.....	7-11
7-10	An Example of Priority-Based Arbitration Algorithm	7-13
8-1	e300c3 Core Block Diagram.....	8-2
8-2	e300 Programming Model—Registers.....	8-16
8-3	e300c3 Data Cache Organization.....	8-30
8-4	Core Interface.....	8-38
9-1	Interrupt Sources Block Diagram for MPC8309	9-3
9-2	System Global Interrupt Configuration Register (SICFR)	9-7
9-3	System Global Interrupt Vector Register (SIVCR).....	9-9
9-4	System Internal Interrupt Pending Register (SIPNR_H).....	9-12
9-5	System Internal Interrupt Pending Register (SIPNR_L).....	9-13
9-6	System Internal Interrupt Group A Priority Register (SIPRR_A).....	9-14
9-7	System Internal Interrupt Group B Priority Register (SIPRR_B).....	9-15
9-8	System Internal Interrupt Group C Priority Register (SIPRR_C).....	9-16
9-9	System Internal Interrupt Group D Priority Register (SIPRR_D).....	9-16
9-10	System Internal Interrupt Mask Register (SIMSR_H).....	9-17
9-11	System Internal Interrupt Mask Register (SIMSR_L)	9-18
9-12	System Internal Interrupt Control Register (SICNR)	9-19
9-13	System External Interrupt Pending Register (SEPNR).....	9-20
9-14	System Mixed Interrupt Group A Priority Register (SMPRR_A).....	9-21
9-15	System Mixed Interrupt Group B Priority Register (SMPRR_B)	9-22
9-16	System External Interrupt Mask Register (SEMSR)	9-23
9-17	System External Interrupt Control Register (SECNR)	9-24
9-18	System Error Status Register (SERSR).....	9-25
9-19	System Error Mask Register (SERMR)	9-26
9-20	System External Interrupt Polarity Control Register (SEPCR)	9-28
9-21	System Internal Interrupt Force Register (SIFCR_H)	9-28
9-22	System Internal Interrupt Force Register (SIFCR_L).....	9-29

Figures

Figure Number	Title	Page Number
9-23	System External Interrupt Force Register (SEFCR)	9-30
9-24	System Error Status Register (SERFR).....	9-30
9-25	System Critical Interrupt Vector Register (SCVCR)	9-31
9-26	System Management Interrupt Vector Register (SMVCR).....	9-32
9-27	QUICC Engine Ports Interrupt Event Register (CEPIER).....	9-33
9-28	QUICC Engine Ports Interrupt Mask Register (CEPIMR).....	9-34
9-29	QUICC Engine Ports Interrupt Control Register (CEPICR)	9-35
9-30	Interrupt Structure for MPC8309	9-36
9-31	DDR Interrupt Request Masking	9-42
10-1	DDR Memory Controller Simplified Block Diagram.....	10-2
10-2	Chip Select Bounds Registers (CS _n _BNDS).....	10-11
10-3	Chip Select Configuration Register (CS _n _CONFIG).....	10-12
10-4	DDR SDRAM Timing Configuration 3 (TIMING_CFG_3)	10-13
10-5	DDR SDRAM Timing Configuration 0 (TIMING_CFG_0)	10-14
10-6	DDR SDRAM Timing Configuration 1 (TIMING_CFG_1).....	10-16
10-7	DDR SDRAM Timing Configuration 2 Register (TIMING_CFG_2).....	10-18
10-8	DDR SDRAM Control Configuration Register (DDR_SDRAM_CFG).....	10-19
10-9	DDR SDRAM Control Configuration Register 2 (DDR_SDRAM_CFG_2).....	10-22
10-10	DDR SDRAM Mode Configuration Register (DDR_SDRAM_MODE).....	10-23
10-11	DDR SDRAM Mode 2 Configuration Register (DDR_SDRAM_MODE_2).....	10-24
10-12	DDR SDRAM Mode Control Register (DDR_SDRAM_MD_CNTL).....	10-25
10-13	DDR SDRAM Interval Configuration Register (DDR_SDRAM_INTERVAL)	10-27
10-14	DDR SDRAM Data Initialization Configuration Register (DDR_DATA_INIT).....	10-28
10-15	DDR SDRAM Clock Control Configuration Register (DDR_SDRAM_CLK_CNTL).....	10-28
10-16	DDR Initialization Address Configuration Register (DDR_INIT_ADDR)	10-29
10-17	DDR IP Block Revision 1 (DDR_IP_REV1)	10-29
10-18	DDR IP Block Revision 2 (DDR_IP_REV2)	10-30
10-19	10-30
10-20	Memory Data Path Error Injection Mask Low Register (DATA_ERR_INJECT_LO).....	10-31
10-21	Memory Data Path Error Injection Mask ECC Register (ERR_INJECT).....	10-31
10-22	Memory Data Path Read Capture High Register (CAPTURE_DATA_HI).....	10-32
10-23	Memory Data Path Read Capture Low Register (CAPTURE_DATA_LO)	10-32
10-24	Memory Data Path Read Capture ECC Register (CAPTURE_ECC).....	10-33
10-25	Memory Error Detect Register (ERR_DETECT)	10-33
10-26	Memory Error Disable Register (ERR_DISABLE).....	10-34
10-27	Memory Error Interrupt Enable Register (ERR_INT_EN).....	10-35
10-28	Memory Error Attributes Capture Register (CAPTURE_ATTRIBUTES).....	10-36
10-29	Memory Error Address Capture Register (CAPTURE_ADDRESS)	10-37
10-30	Single-Bit ECC Memory Error Management Register (ERR_SBE)	10-37
10-31	DDR Memory Controller Block Diagram	10-39
10-32	Typical Dual Data Rate SDRAM Internal Organization.....	10-40

Figures

Figure Number	Title	Page Number
10-33	Typical DDR SDRAM Interface Signals	10-40
10-34	Example 64-Mbyte DDR SDRAM Configuration With ECC	10-41
10-35	DDR SDRAM Burst Read Timing—ACTTORW = 3, MCAS Latency = 2	10-49
10-36	DDR SDRAM Single-Beat (Double Word) Write Timing—ACTTOR	10-49
10-37	DDR SDRAM Single-Beat (Double Word) Write Timing—ACTTORW = 3	10-50
10-38	DDR SDRAM Clock Distribution Example for ×8 DDR SDRAMs	10-50
10-39	DDR SDRAM Mode-Set Command Timing	10-51
10-40	Registered DDR SDRAM DIMM Burst Write Timing	10-52
10-41	Write Timing Adjustments Example for Write Latency = 1	10-53
10-42	DDR SDRAM Bank Staggered Auto Refresh Timing	10-54
10-43	DDR SDRAM Power-Down Mode	10-55
10-44	DDR SDRAM Self-Refresh Entry Timing	10-56
10-45	DDR SDRAM Self-Refresh Exit Timing	10-56
11-1	Enhanced Local Bus Controller Block Diagram	11-1
11-2	Base Registers (BR _n)	11-10
11-3	Option Registers (OR _n) in GPCM Mode	11-13
11-4	Option Registers (OR _n) in FCM Mode	11-15
11-5	Option Registers (OR _n) in UPM Mode	11-18
11-6	UPM Memory Address Register (MAR)	11-19
11-7	UPM Mode Registers (MxMR)	11-20
11-8	Memory Refresh Timer Prescaler Register (MRTPR)	11-22
11-9	UPM Data Register in UPM Mode (MDR)	11-23
11-10	FCM Data Register in FCM Mode (MDR)	11-23
11-11	Special Operation Initiation Register (LSOR)	11-24
11-12	UPM Refresh Timer (LURT)	11-24
11-13	Transfer Error Status Register (LTESR)	11-25
11-14	Transfer Error Check Disable Register (LTEDR)	11-27
11-15	Transfer Error Interrupt Enable Register (LTEIR)	11-28
11-16	Transfer Error Attributes Register (LTEATR)	11-29
11-17	Transfer Error Address Register (LTEAR)	11-30
11-18	Local Bus Configuration Register	11-30
11-19	Clock Ratio Register (LCRR)	11-32
11-20	Flash Mode Register	11-33
11-21	Flash Instruction Register	11-35
11-22	Flash Command Register	11-35
11-23	Flash Block Address Register	11-36
11-24	Flash Page Address Register, Small Page Device (ORx[PGS] = 0)	11-36
11-25	Flash Page Address Register, Large Page Device (ORx[PGS] = 1)	11-37
11-26	Flash Byte Count Register	11-38
11-27	Basic Operation of Memory Controllers in the eLBC	11-39
11-28	Example of 8-Bit GPCM Writing 32 Bytes to Address 0x5420 (LCRR[PBYP] = 0)	11-41

Figures

Figure Number	Title	Page Number
11-29	Basic eLBC Bus Cycle with LALE, TA, and $\overline{\text{LCSn}}$	11-42
11-30	Enhanced Local Bus to GPCM Device Interface.....	11-43
11-31	GPCM Basic Read Timing (XACS = 0, ACS = 1x, TRLX = 0, CLKDIV = 4,8).....	11-44
11-32	GPCM General Read Timing Parameters	11-44
11-33	GPCM General Write Timing Parameters	11-46
11-34	GPCM Basic Write Timing (XACS = 0, ACS = 00, CSNT = 1, SCY = 1, TRLX = 0, CLKDIV = 4, 8).....	11-48
11-35	GPCM Relaxed Timing Back-to-Back Reads (XACS = 0, ACS = 1x, SCY = 1, CSNT = 0, TRLX = 1, EHTR = 0, CLKDIV = 4, 8)	11-50
11-36	GPCM Relaxed Timing Back-to-Back Writes (XACS = 0, ACS = 1x, SCY = 0, CSNT = 0, TRLX = 1, CLKDIV = 4, 8).....	11-50
11-37	GPCM Relaxed Timing Write (XACS = 0, ACS = 10, SCY = 0, CSNT = 1, TRLX = 1, CLKDIV = 4, 8).....	11-51
11-38	GPCM Relaxed Timing Write (XACS = 0, ACS = 00, SCY = 1, CSNT = 1, TRLX = 1, CLKDIV = 4, 8).....	11-51
11-39	GPCM Read Followed by Read (TRLX = 0, EHTR = 0, Fastest Timing).....	11-52
11-40	GPCM Read Followed by Write (TRLX = 0, EHTR = 1, One-Cycle Extended Hold Time on Reads)	11-53
11-41	External Termination of GPCM Access(PLL Bypass Mode).....	11-54
11-42	Local Bus to 8-Bit FCM Device Interface	11-56
11-43	FCM Basic Page Read Timing (PGS = 1, CSCT = 0, CST = 0, CHT = 1, RST = 1, SCY = 0, TRLX = 0, EHTR = 1)	11-57
11-44	FCM Buffer RAM Memory Map for Small-Page (512-byte page) NAND Flash Devices	11-59
11-45	FCM Buffer RAM Memory Map for Large-Page (2-Kbyte page) NAND Flash Devices .	11-60
11-46	FCM ECC Calculation	11-60
11-47	ECC Placement in NAND Flash Spare Regions in Relation to FMR[ECCM]	11-61
11-48	FCM Instruction Sequencer Mechanism.....	11-62
11-49	Timing of FCM Command/Address and Write Data Cycles (for TRLX = 0, CHT = 0, CST = 1, SCY = 1, CLKDIV = 4*N).....	11-65
11-50	Example of FCM Command and Address Timing with Minimum Delay Parameters (for TRLX = 0, CHT = 0, CST = 0, SCY = 0, CLKDIV = 4*N).....	11-66
11-51	Example of FCM Command and Address Timing with Relaxed Parameters (for TRLX = 1, CHT = 0, CST = 1, SCY = 2, CLKDIV = 4*N).....	11-66
11-52	FCM Delay Prior to Sampling LFRB State	11-67
11-53	FCM Read Data Timing (for TRLX = 0, RST = 0, SCY = 1, CLKDIV = 4*N).....	11-67
11-54	FCM Read Data Timing with Extended Hold Time (for TRLX = 0, EHTR = 1, RST = 1, SCY = 1, CLKDIV = 4*N)	11-68
11-55	FCM Buffer RAM Memory Map During Boot Loading	11-70
11-56	User-Programmable Machine Functional Block Diagram.....	11-71
11-57	RAM Array Indexing	11-72

Figures

Figure Number	Title	Page Number
11-58	Memory Refresh Timer Request Block Diagram	11-73
11-59	UPM Clock Scheme for LCRR[CLKDIV] = 2.....	11-77
11-60	UPM Clock Scheme for LCRR[CLKDIV] = 4 or 8	11-77
11-61	RAM Array and Signal Generation	11-77
11-62	RAM Word Fields	11-78
11-63	LCSn Signal Selection	11-81
11-64	LBS Signal Selection	11-82
11-65	UPM Read Access Data Sampling.....	11-85
11-66	Effect of LUPWAIT Signal.....	11-86
11-67	Multiplexed Address/Data Bus for 26-Bit Addressing	11-87
11-68	Local Bus Peripheral Hierarchy for High Bus Speeds.....	11-88
11-69	GPCM Address Timings	11-88
11-70	GPCM Data Timings.....	11-89
11-71	Interface to Different Port-Size Devices	11-90
11-72	Single-Beat Read Access to FPM DRAM	11-96
11-73	Single-Beat Write Access to FPM DRAM	11-97
11-74	Burst Read Access to FPM DRAM Using LOOP (Two Beats Shown).....	11-98
11-75	Refresh Cycle (CBR) to FPM DRAM	11-99
11-76	Exception Cycle	11-100
11-77	Interface to ZBT SRAM	11-101
12-1	System Connection of the eSDHC	12-2
12-2	eSDHC Block Diagram.....	12-3
12-3	DMA System Address Register (DSADDR)	12-7
12-4	Block Attributes Register (BLKATTR)	12-7
12-5	Command Argument Register (CMDARG)	12-8
12-6	Transfer Type Register (XFERTYP).....	12-9
12-7	Command Response 0–3 Register (CMDRSPn)	12-12
12-8	Buffer Data Port Register (DATPORT)	12-14
12-9	Present State Register (PRSSTAT).....	12-15
12-10	Protocol Control Register (PROCTL).....	12-19
12-11	System Control Register (SYSCTL).....	12-22
12-12	Interrupt Status Register (IRQSTAT).....	12-25
12-13	Interrupt Status Enable Register (IRQSTATEN)	12-29
12-14	Interrupt Signal Enable Register (IRQSIGEN).....	12-31
12-15	Auto CMD12 Error Status Register (AUTOC12ERR)	12-33
12-16	Host Capabilities Register (HOSTCAPBLT).....	12-35
12-17	Watermark Level Register (WML)	12-36
12-18	Force Event Register (FEVT)	12-37
12-19	Host Controller Version Register (HOSTVER).....	12-39
12-20	eSDHC Buffer Scheme	12-40
12-21	Example of Dividing a Large Data Transfer	12-42

Figures

Figure Number	Title	Page Number
12-22	DMA CSB Interface Block	12-43
12-23	Command CRC Shift Register	12-44
12-24	Two Stages of Clock Divider	12-45
12-25	a) Card Interrupt Scheme; b) Card Interrupt Detection and Handling Procedure	12-47
12-26	Flow Diagram for Card Detection	12-49
12-27	Flow Chart for Reset of eSDHC and SD I/O Card	12-50
13-1	DMA Block Diagram.....	13-1
13-2	DMA Control Register (DMACR)	13-4
13-3	DMA Error Status Register (DMAES)	13-6
13-4	DMA Enable Request Register (DMAERQ)	13-8
13-5	DMA Enable Error Interrupt Register (DMAEEI)	13-9
13-6	DMA Set Enable Error Interrupt Register	13-9
13-7	DMA Clear Enable Error Interrupt Register.....	13-10
13-8	DMA Clear Interrupt Request Register	13-11
13-9	DMA Clear Error Register.....	13-11
13-10	DMA Set START Bit Register	13-12
13-11	DMA Clear DONE Status Register.....	13-13
13-12	DMA Interrupt Request Register Low (DMAINT)	13-14
13-13	DMA Error Register (DMAERR).....	13-15
13-14	DMA General Purpose Output Register (DMAGPOR).....	13-15
13-15	DMA Clear DONE Status Register.....	13-17
13-16	TCD Word 0 (TCDn.saddr) Field	13-18
13-17	TCD Word 1 (TCDn.{soff, smod, ssize, dmod, dsize}) Fields.....	13-19
13-18	TCD Word 2 (TCDn.nbytes) Field.....	13-20
13-19	TCD Word 3 (TCDn.slast) Field.....	13-20
13-20	TCD Word 4 (TCDn.daddr) Field.....	13-21
13-21	TCD Word 5 (TCDn.{citer, doff}) Fields	13-21
13-22	TCD Word 6 (TCDn.dlast_sga) Field	13-22
13-23	TCD Word 7 (TCDn.{biter, control/status}) Fields	13-23
13-24	DMA Operation—Part 1	13-27
13-25	DMA Operation—Part 2	13-28
13-26	DMA Operation—Part 3	13-29
14-1	DMA/Messaging Unit Block Diagram	14-1
14-2	Outbound Message Interrupt Status Register (OMISR)	14-4
14-3	Outbound Message Interrupt Mask Register (OMIMR).....	14-4
14-4	Inbound Message Registers (IMR0, IMR1).....	14-5
14-5	Outbound Message Registers (OMR0–OMR1).....	14-5
14-6	Outbound Doorbell Register (ODR)	14-6
14-7	Inbound Doorbell Register (IDR)	14-7
14-8	Inbound Message Interrupt Status Register (IMISR).....	14-7
14-9	Inbound Message Interrupt Mask Register (IMIMR)	14-8

Figures

Figure Number	Title	Page Number
14-10	DMA Mode Register (DMAMR _n)	14-9
14-11	DMA Status Register (DMASR _n)	14-11
14-12	DMA Current Descriptor Address Register (DMACDAR _n).....	14-12
14-13	DMA Source Address Register (DMASAR _n)	14-13
14-14	DMA Destination Address Register (DMADAR _n)	14-13
14-15	DMA Byte Count Register (DMABCR _n).....	14-14
14-16	DMA Next Descriptor Address Register (DMANDAR _n).....	14-14
14-17	DMA General Status Register (DMAGSR).....	14-15
14-18	DMA Controller Block Diagram	14-16
14-19	DMA Chain of Segment Descriptors	14-19
15-1	FlexCAN Block Diagram.....	15-2
15-2	Message Buffer Structure.....	15-7
15-3	Rx FIFO Structure.....	15-10
15-4	ID Table (0–7)	15-11
15-5	Module Configuration Register (MCR).....	15-12
15-6	Control Register (CTRL)	15-16
15-7	Free Running Timer (TIMER)	15-18
15-8	Rx Global Mask Register (RXGMASK)	15-19
15-9	Rx 14 Mask Register.....	15-20
15-10	Rx 15 Mask Register.....	15-20
15-11	Error Counter Register (ECR).....	15-21
15-12	Error and Status Register (ESR)	15-22
15-13	Interrupt Masks 2 Register (IMASK2)	15-24
15-14	Interrupt Masks 1 Register (IMASK1)	15-25
15-15	Interrupt Flags 2 Register (IFLAG2)	15-26
15-16	Interrupt Flags 1 Register (IFLAG1)	15-27
15-17	Rx Individual Mask Registers (RXIMR0–RXIMR63).....	15-28
15-18	CAN Engine Clocking Scheme.....	15-37
15-19	Segments within the Bit Time.....	15-38
15-20	Arbitration, Match, and Move Time Windows	15-39
16-1	USB Interface Block Diagram	16-1
16-2	Capability Registers Length (CAPLENGTH)	16-6
16-3	Host Controller Interface Version (HCIVERSION)	16-7
16-4	Host Controller Structural Parameters (HCSPARAMS).....	16-7
16-5	Host Control Capability Parameters (HCCPARAMS)	16-8
16-6	Device Interface Version (DCIVERSION)	16-9
16-7	Device Control Capability Parameters (DCCPARAMS).....	16-9
16-8	USB Command Register (USBCMD)	16-10
16-9	USB Status Register (USBSTS).....	16-13
16-10	USB Interrupt Enable (USBINTR).....	16-15
16-11	USB Frame Index (FRINDEX).....	16-17

Figures

Figure Number	Title	Page Number
16-12	Periodic Frame List Base Address (PERIODICLISTBASE)	16-18
16-13	Device Address (DEVICEADDR).....	16-19
16-14	Current Asynchronous List Address (ASYNCLISTADDR)	16-19
16-15	Endpoint List Address (ENDPOINTLISTADDR).....	16-20
16-16	Master Interface Data Burst Size (BURSTSIZE)	16-20
16-17	Transmit FIFO Tuning Controls (TXFILLTUNING)	16-21
16-18	ULPI Register Access (ULPI VIEWPORT)	16-23
16-19	Configure Flag Register (CONFIGFLAG)	16-24
16-20	Port Status and Control (PORTSC).....	16-25
16-21	OTG Status Control (OTGSC).....	16-30
16-22	USB Mode (USBMODE)	16-32
16-23	Endpoint Setup Status (ENDPTSETUPSTAT)	16-33
16-24	Endpoint Initialization (ENDPTPRIME).....	16-33
16-25	Endpoint Flush (ENDPTFLUSH)	16-34
16-26	Endpoint Status (ENDPTSTATUS)	16-35
16-27	Endpoint Complete (ENDPTCOMPLETE).....	16-36
16-28	Endpoint Control 0 (ENDPTCTRL0)	16-36
16-29	Endpoint Control 1 to 5 (ENDPTCTRL _n).....	16-37
16-30	Snoop 1 and Snoop 2 (SNOOP _n).....	16-39
16-31	Age Count Threshold (AGE_CNT_THRESH).....	16-41
16-32	Priority Control (PRI_CTRL)	16-42
16-33	System Interface Control Register (SI_CTRL).....	16-42
16-34	USB General-Purpose Register (CONTROL)	16-43
16-35	Periodic Schedule Organization	16-46
16-36	Frame List Link Pointer Format.....	16-46
16-37	Asynchronous Schedule Organization	16-47
16-38	Isochronous Transaction Descriptor (iT _D)	16-48
16-39	Split-Transaction Isochronous Transaction Descriptor (si _{TD})	16-51
16-40	Queue Element Transfer Descriptor (q _{TD}).....	16-55
16-41	Queue Head Layout	16-60
16-42	Frame Span Traversal Node Structure	16-64
16-43	Derivation of Pointer into Frame List Array.....	16-70
16-44	General Format of Asynchronous Schedule List	16-70
16-45	Frame Boundary Relationship Between HS Bus and FS/LS Bus	16-71
16-46	Relationship of Periodic Schedule Frame Boundaries to Bus Frame Boundaries	16-72
16-47	Example Periodic Schedule	16-74
16-48	Example Association of iT _D s to Client Request Buffer	16-77
16-49	Generic Queue Head Unlink Scenario	16-82
16-50	Asynchronous Schedule List with Annotation to Mark Head of List.....	16-83
16-51	Example Mapping of q _{TD} Buffer Pointers to Buffer Pages	16-85
16-52	Host Controller Asynchronous Schedule Split-Transaction State Machine	16-88

Figures

Figure Number	Title	Page Number
16-53	Split Transaction, Interrupt Scheduling Boundary Conditions	16-91
16-54	General Structure of EHCI Periodic Schedule Utilizing Interrupt Spreading	16-92
16-55	Example Host Controller Traversal of Recovery Path via FSTNs.....	16-94
16-56	Split Transaction State Machine for Interrupt.....	16-97
16-57	Split Transaction, Isochronous Scheduling Boundary Conditions	16-104
16-58	siTD Scheduling Boundary Examples	16-106
16-59	Split Transaction State Machine for Isochronous	16-109
16-60	Endpoint Queue Head Organization	16-122
16-61	Endpoint Queue Head Layout.....	16-122
16-62	Endpoint Transfer Descriptor (dTD).....	16-125
16-63	USB 2.0 Device States	16-129
16-64	Endpoint Queue Head Diagram	16-141
16-65	Software Link Pointers.....	16-143
17-1	I ² C Block Diagram.....	17-1
17-2	I ² C _n Address Register (I2C _n ADR).....	17-5
17-3	I ² C _n Frequency Divider Register (I2C _n FDR)	17-6
17-4	I ² C _n Control Register (I2C _n CR).....	17-7
17-5	I ² C _n Status Register (I2C _n SR)	17-8
17-6	I ² C _n Data Register (I2C _n DR).....	17-9
17-7	I ² C _n Digital Filter Sampling Rate Register (I2C _n DFSRR).....	17-10
17-8	I ² C 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 ² C 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, UTHR3 and UTHR4)	18-6
18-4	Divisor Most Significant Byte Registers (UDMB1 and UDMB2, UDMB3 and UDMB4) .	18-7
18-5	Divisor Least Significant Byte Registers (UDLB1 and UDLB2, UDLB3 and UDLB4)	18-7
18-6	Interrupt Enable Registers (UIER1 and UIER2, UIER3 and UIER4)	18-8
18-7	Interrupt ID Registers (UIIR1 and UIIR2, UIIR3 and UIIR4)	18-9
18-8	FIFO Control Registers (UFCR1 and UFCR2, UFCR3 and UFCR4).....	18-11
18-9	Alternate Function Register (UAFR).....	18-11
18-10	Line Control Register (ULCR1 and ULCR2, ULCR3 and ULCR4).....	18-12
18-11	Modem Control Register (UMCR1 and UMCR2, UMCR3 and UMCR4)	18-14
18-12	Line Status Register (ULSR1 and ULSR2, ULSR3 and ULSR4)	18-15
18-13	Modem Status Register (UMSR1 and UMSR2, UMSR3 and UMSR4).....	18-16
18-14	Scratch Register (USCR)	18-17
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

Figures

Figure Number	Title	Page Number
19-2	Single-Master/Multi-Slave Configuration	19-4
19-3	Multiple-Master Configuration	19-6
19-4	SPMODE-SPI Mode Register Definition	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 Definition.....	19-12
19-8	SPIM—SPI Mask Register Definition	19-13
19-9	SPI Command Register Definition	19-14
19-10	SPI Transmit Data Hold Register Definition	19-14
19-11	SPI Receive Data Hold Register Definition.....	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–GP ₂ DAT)	21-4
21-5	GPIO _n Interrupt Event Register (GP ₁ IER–GP ₂ IER).....	21-5
21-6	GPIO _n Interrupt Mask Register (GP ₁ IMR–GP ₂ IMR)	21-5
21-7	GPIO _n Interrupt Control Register (GP ₁ ICR–GP ₂ ICR).....	21-6
22-1	I/O Sequencer Block Diagram	22-1
22-2	PCI Outbound Translation Address Registers (POTAR _n).....	22-3
22-3	PCI Outbound Base Address Registers (POBAR _n).....	22-3
22-4	PCI Outbound Comparison Mask Registers (POCMR _n)	22-4
22-5	Power Management Control Register (PMCR)	22-5
22-6	Discard Timer Control Register (DTCR).....	22-6
22-7	Outbound PCI Memory Address Translation	22-8
23-1	PCI Controller Block Diagram	23-2
23-2	PCI Interface External Signals.....	23-5
23-3	PCI_CONFIG_ADDRESS Register	23-12
23-4	PCI_CONFIG_DATA	23-15
23-5	PCI Error Status Register (PCI_ESR).....	23-15
23-6	PCI Error Capture Disable Register (PCI_ECDR)	23-16
23-7	PCI Error Enable Register (PCI_EER)	23-17
23-8	PCI Error Attributes Capture Register (PCI_EATCR)	23-18
23-9	PCI Error Address Capture Register (PCI_EACR)	23-19
23-10	PCI Error Extended Address Capture Register (PCI_EEACR).....	23-20
23-11	PCI Error Data Capture Register (PCI_EDCR).....	23-20
23-12	PCI General Control Register (PCI_GCR)	23-21

Figures

Figure Number	Title	Page Number
23-13	PCI Error Control Register (PCI_ECR).....	23-21
23-14	PCI General Status Register (PCI_GSR).....	23-22
23-15	PCI Inbound Translation Address Registers (PITARn).....	23-23
23-16	PCI Inbound Base Address Registers (PIBARn).....	23-23
23-17	PCI Inbound Extended Base Address Registers (PIEBARn).....	23-24
23-18	PCI Inbound Window Attribute Registers (PIWARn).....	23-24
23-19	Vendor ID Configuration Register.....	23-26
23-20	Device ID Configuration Register.....	23-27
23-21	PCI Command Configuration Register.....	23-27
23-22	PCI Status Configuration Register.....	23-28
23-23	Revision ID Configuration Register.....	23-29
23-24	Standard Programming Interface Configuration Register.....	23-30
23-25	Subclass Code Configuration Register.....	23-30
23-26	Base Class Code Configuration Register.....	23-31
23-27	Cache Line Size Configuration Register.....	23-31
23-28	Latency Timer Configuration Register.....	23-32
23-29	Header Type Configuration Register.....	23-32
23-30	BIST Control Configuration Register.....	23-32
23-31	PIMMR Base Address Configuration Register.....	23-33
23-32	GPL Base Address Register 0.....	23-33
23-33	GPL Base Address Registers 1–2.....	23-34
23-34	GPL Extended Base Address Registers 1–2.....	23-35
23-35	Subsystem Vendor ID Configuration Register.....	23-35
23-36	Subsystem Device ID Configuration Register.....	23-36
23-37	Capabilities Pointer Configuration Register.....	23-36
23-38	Interrupt Line Configuration Register.....	23-36
23-39	Interrupt Pin Register.....	23-37
23-40	Minimum Grant Configuration Register.....	23-37
23-41	Maximum Latency Configuration Register.....	23-37
23-42	PCI Function Configuration Register.....	23-38
23-43	PCI Arbiter Control Register (PCIACR).....	23-39
23-44	Hot Swap Register Block.....	23-40
23-45	PCI Arbitration Example.....	23-42
23-46	Single Beat Read Example.....	23-47
23-47	Burst Read Example.....	23-47
23-48	Single Beat Write Example.....	23-48
23-49	Burst Write Example.....	23-48
23-50	Target-Initiated Terminations.....	23-49
23-51	PCI Parity Operation.....	23-55
23-52	Inbound PCI Memory Address Translation.....	23-56
23-53	Address Invariant Byte Ordering—4 bytes Outbound.....	23-57

Figures

Figure Number	Title	Page Number
23-54	Address Invariant Byte Ordering—4 bytes Inbound	23-58
23-55	Address Invariant Byte Ordering—8 bytes Outbound.....	23-58
23-56	Address Invariant Byte Ordering—2 bytes Inbound	23-58
23-57	CFG_DATA Byte Ordering.....	23-59
24-1	QUICC Engine Block Architectural Block Diagram for the MPC8309.....	24-2
24-2	Data Paths	24-5
24-3	Serial DMA Status Register (SDSR)	24-6
24-4	Serial DMA Mode Register (SDMR)	24-7
24-5	DMA Read Data Path	24-9
24-6	DMA Write Data Path.....	24-9
24-7	DMA Command Queue	24-10
24-8	Serial DMA Threshold Register (SDTR).....	24-10
24-9	Serial DMA Hysteresis Register (SDHY)	24-11
24-10	Serial DMA Transfer Address Register (SDTA)	24-11
24-11	Serial DMA Transfer MSNUM Register (SDTM)	24-12
24-12	Serial DMA Temporary Buffer Base in Multi-User RAM Value (SDEBCR).....	24-12
24-13	QUICC Engine Module Interrupt Structure	24-13
24-14	Interrupt Request Masking.....	24-19
24-15	QUICC Engine System Interrupt Configuration Register (CICR)	24-22
24-16	QUICC Engine System Internal Interrupt Control Register (CICNR)	24-23
24-17	QUICC Engine System RISC Interrupts Control Register (CRICR)	24-24
24-18	QUICC Engine System Interrupt Priority Register for WCC (CIPWCC).....	24-25
24-19	QUICC Engine System Interrupt Priority Register for XCC (CIPXCC).....	24-26
24-20	QUICC Engine System Interrupt Priority Register for YCC (CIPYCC).....	24-27
24-21	QUICC Engine System Interrupt Priority Register for ZCC (CIPZCC)	24-28
24-22	QUICC Engine System Interrupt Priority Register for RISC Tasks A (CIPRTA).....	24-29
24-23	QUICC Engine System Interrupt Priority Register for RISC Tasks B (CIPRTB).....	24-29
24-24	QUICC Engine System Interrupt Pending Register (CIPNR)	24-30
24-25	CIMR Field	24-31
24-26	CRIPNR Fields	24-32
24-27	CRIMR Fields	24-33
24-28	QUICC Engine System Interrupt Vector Register (CIVEC).....	24-33
24-29	Interrupt Table Handling Example.....	24-34
24-30	QUICC Engine High System Interrupt Vector Register (CHIVEC).....	24-35
24-31	I-RAM Ready Register (IRReady)	24-36
24-32	External Virtual Thread Event/Mask Register (CEVTxx_ER/CEVTxxMR)	24-39
24-33	Virtual Thread Pending Event/Mask Register (CEVTPER/CEVTPMR)	24-40
24-34	QUICC Engine Multiplexing and Timers Logic (CMX) Block Diagram	24-42
24-35	CMXUCR1[HBM1] and CMXUCR1[HBM3] location on MPC8309	24-43

Tables

Table Number	Title	Page Number
1-1	QUICC Engine Protocols.....	1-13
1-2	UCC Protocol Enablement in the QUICC Engine Block	1-14
2-1	IMMR Memory Map	2-2
3-1	MPC8309 Signal Reference by Functional Block	3-4
3-2	Output Signal States During System Reset.....	3-20
4-1	System Control Signals.....	4-1
4-2	External Clock Signals.....	4-2
4-3	Reset Causes	4-4
4-4	Reset Actions	4-5
4-5	Reset Configuration Words Source.....	4-9
4-6	Selecting Reset Configuration Input Signals	4-10
4-7	RCWLR Bit Settings.....	4-11
4-8	System PLL VCO Division.....	4-12
4-9	System PLL Ratio	4-13
4-10	QUICC Engine PLL Multiplication Factor.....	4-13
4-11	Reset Configuration Word High Bit Settings.....	4-14
4-12	Boot Memory Space.....	4-16
4-13	Boot Sequencer Configuration	4-16
4-14	Boot ROM Location.....	4-17
4-15	e300 Core True Little-Endian	4-17
4-16	Local Bus Configuration EEPROM Addresses	4-18
4-17	Local Bus Reset Configuration Words Data Structure.....	4-18
4-18	Local Bus Controller Setting when Loading RCW	4-19
4-19	RCW Values Corresponding to Hard Coded Options.....	4-23
4-20	Hard Coded Reset Configuration Word Low Fields Values	4-23
4-21	Hard-Coded Reset Configuration Word High Field Values.....	4-24
4-22	Configurable Clock Units	4-26
4-23	Reset Configuration and Status Registers Memory Map.....	4-27
4-24	Reset Status Register Field Descriptions	4-28
4-25	RMR Field Descriptions	4-29
4-26	RPR Bit Descriptions	4-30
4-27	RCR Bit Settings.....	4-31
4-28	RCER Bit Settings	4-31
4-29	Clock Configuration Registers Memory Map.....	4-31
4-30	System PLL Mode Register Bit Settings	4-32
4-31	OCCR Bit Settings	4-33
4-32	SCCR Bit Descriptions	4-34
5-1	Boot Source Selection for On-Chip ROM Boot- RCWHR Settings	5-1
5-2	SD/MMC Card Data Structure.....	5-4
5-3	Config Address Field Description, CNT = 0	5-5
5-4	Config Address Field Description, CNT = 1	5-6

Tables

Table Number	Title	Page Number
5-5	SPI EEPROM Data Structure.....	5-11
5-6	Config Address Field Description, CNT = 0	5-13
5-7	Config Address Field Description, CNT = 1	5-13
6-1	Local Access Windows Target Interface.....	6-1
6-2	Local Access Windows Example.....	6-2
6-3	Format of Window Definitions	6-3
6-4	Local Access Register Memory Map.....	6-5
6-5	IMMRBAR Bit Settings.....	6-7
6-6	ALTCBAR Bit Settings.....	6-8
6-7	LBLAWBAR0–LBLAWBAR3 Bit Settings.....	6-8
6-8	LBLAWBAR0[BASE_ADDR] Reset Value	6-8
6-9	LBLAWAR0–LBLAWAR3 Bit Settings.....	6-9
6-10	LBLAWAR0[EN] Reset Value.....	6-10
6-11	PCILAWBAR0–PCILAWBAR1 Bit Settings	6-10
6-12	PCILAWBAR0[BASE_ADDR] Reset Value	6-11
6-13	PCILAWAR0–PCILAWAR1 Bit Settings.....	6-11
6-14	PCILAWAR0[EN] Reset Value.....	6-12
6-15	DDRLAWBAR0–DDRLAWBAR1 Bit Settings.....	6-12
6-16	DDRLAWBAR0[BASE_ADDR] Reset Value	6-12
6-17	DDRLAWAR0–DDRLAWAR1 Bit Settings	6-13
6-18	DDRLAWAR0[EN] Reset Value	6-14
6-19	Overlapping Local Access Windows	6-14
6-20	System Configuration Register Memory Map.....	6-16
6-21	SGPRL Bit Settings	6-18
6-22	SGPRH Bit Settings	6-18
6-23	SPRIDR Bit Settings.....	6-19
6-24	PARTID Coding	6-19
6-25	REVID Coding	6-19
6-26	SPCR Bit Settings	6-20
6-27	SICR_1 Bit Settings for MPC8309.....	6-22
6-28	SICR_2 Bit Settings for MPC8309.....	6-26
6-29	QE UART Multiplexing Details	6-27
6-30	DDRCDR Field Descriptions.....	6-29
6-31	DDRDSR Field Descriptions	6-30
6-32	SDHCCR Field Description.....	6-31
6-33	CAN_DBG_CTRL Field Descriptions	6-33
6-34	SPI_CS Field Descriptions.....	6-33
6-35	GPR_1 Field Descriptions	6-34
6-36	CAN_INT_STAT Field Descriptions.....	6-36
6-37	DUART_INT_STAT Field Descriptions.....	6-38
6-38	GPIO_INT_STAT Field Descriptions	6-39

Tables

Table Number	Title	Page Number
6-39	WDT Register Address Map.....	6-41
6-40	SWCRR Bit Settings.....	6-42
6-41	SWCNR Bit Settings.....	6-43
6-42	SWSRR Bit Settings.....	6-44
6-43	RTC External Signals.....	6-48
6-44	RTC Register Address Map.....	6-48
6-45	RTCNR Bit Settings.....	6-49
6-46	RTLDR Bit Settings.....	6-50
6-47	RTPSR Bit Settings.....	6-50
6-48	RTCTR Bit Settings.....	6-51
6-49	RTEVR Bit Settings.....	6-51
6-50	RTALR Bit Settings.....	6-52
6-51	PIT Signal Properties.....	6-55
6-52	PIT External Signal—Detailed Signal Descriptions.....	6-55
6-53	PIT Register Address Map.....	6-55
6-54	PTCNR Bit Settings.....	6-56
6-55	PTLDR Bit Settings.....	6-57
6-56	PTPSR Bit Settings.....	6-57
6-57	PTCTR Bit Settings.....	6-57
6-58	PTEVR Bit Settings.....	6-58
6-59	GTM Signal Properties.....	6-62
6-60	GTM External Signals—Detailed Signal Descriptions.....	6-63
6-61	GTM Register Address Map.....	6-64
6-62	GTCFR1 Bit Settings.....	6-66
6-63	GTCFR2 Bit Settings.....	6-68
6-64	GTMDR Bit Settings.....	6-69
6-65	GTRFR Bit Settings.....	6-70
6-66	GTCPR _n Bit Settings.....	6-71
6-67	GTCNR Bit Settings.....	6-71
6-68	GTEVR _n Bit Settings.....	6-72
6-69	GTPSR _n Bit Settings.....	6-72
6-70	System Control Signals—Detailed Signal Descriptions.....	6-77
6-71	Power Management Controller Registers Memory Map.....	6-77
6-72	PMCCR Bit Settings.....	6-78
6-73	Software-Controller Power-Down States—Basic Description.....	6-78
7-1	Arbiter Register Map.....	7-2
7-2	ACR Field Descriptions.....	7-3
7-3	ATR Field Descriptions.....	7-5
7-4	AER Field Descriptions.....	7-5
7-5	AIDR Field Descriptions.....	7-6
7-6	AMR Field Descriptions.....	7-7

Tables

Table Number	Title	Page Number
7-7	AEATR Field Descriptions	7-8
7-8	AEADR Field Descriptions	7-10
7-9	AERR Field Descriptions.....	7-10
7-10	Address Only Transaction Type Encoding.....	7-15
7-11	Reserved Transaction Type Encoding.....	7-16
7-12	Illegal Transaction Type Encoding	7-16
8-1	Device Revision Level Cross-Reference	8-13
8-2	MSR Bit Descriptions	8-18
8-3	e300 HID0 Bit Descriptions.....	8-22
8-4	Using HID0[ECLK] and HID0[SBCLK] to Configure <i>clk_out</i>	8-24
8-5	HID1 Bit Descriptions	8-25
8-6	e300HID2 Bit Descriptions.....	8-25
8-7	Interrupt Classifications	8-33
8-8	Exceptions and Interrupts.....	8-33
8-9	Differences Between e300 and G2_LE Cores	8-40
9-1	IPIC Signal Properties.....	9-5
9-2	IPIC External Signals—Detailed Signal Descriptions.....	9-5
9-3	IPIC Register Address Map	9-6
9-4	QUICC Engine Ports Interrupts Register Address Map	9-7
9-5	SICFR Field Descriptions	9-7
9-6	SIVCR Field Descriptions	9-9
9-7	IVEC/CVEC/MVEC Field Definition	9-10
9-8	SIPNR_H/SIFCR_H/SIMSR_H Bit Assignments.....	9-12
9-9	SIPNR_H Field Descriptions	9-12
9-10	SIPNR_L/SIFCR_L/SIMSR_L Bit Assignments	9-13
9-11	SIPNR_L Field Descriptions	9-14
9-12	SIPRR_A Field Descriptions	9-14
9-13	SIPRR_B Field Descriptions	9-15
9-14	SIPRR_C Field Descriptions	9-16
9-15	SIPRR_D Field Descriptions	9-17
9-16	SIMSR_H Field Descriptions	9-18
9-17	SIMSR_L Field Descriptions.....	9-18
9-18	SICNR Field Descriptions	9-19
9-19	SEPNR Field Descriptions.....	9-21
9-20	SMPRR_A Field Descriptions	9-21
9-21	SMPRR_B Field Descriptions	9-22
9-22	SEMSR Field Descriptions	9-23
9-23	SECNR Field Descriptions	9-24
9-24	SERSR/SERM/SERFR Bit Assignments.....	9-25
9-25	SERSR Field Descriptions	9-25
9-26	SERM Field Descriptions.....	9-26

Tables

Table Number	Title	Page Number
9-27	SERCR Field Descriptions.....	9-27
9-28	SEPCR Field Descriptions	9-28
9-29	SIFCR_H Field Descriptions	9-29
9-30	SIFCR_L Field Descriptions.....	9-29
9-31	SEFCR Field Descriptions	9-30
9-32	SERFR Field Descriptions	9-31
9-33	SCVCR Field Descriptions	9-31
9-34	SMVCR Field Descriptions	9-32
9-35	CEPIER Bit Settings	9-33
9-36	CEPIMR Bit Settings	9-34
9-37	CEPICR Bit Settings.....	9-35
9-38	Interrupt Source Priority Levels.....	9-38
10-1	DDR Memory Interface Signal Summary	10-3
10-2	Memory Address Signal Mappings.....	10-5
10-3	Memory Interface Signals—Detailed Signal Descriptions.....	10-6
10-4	Clock Signals—Detailed Signal Descriptions	10-8
10-5	DDR Memory Controller Memory Map.....	10-9
10-6	CS _n _BNDS Field Descriptions.....	10-11
10-7	CS _n _CONFIG Field Descriptions	10-12
10-8	TIMING_CFG_3 Field Descriptions	10-13
10-9	TIMING_CFG_0 Field Descriptions	10-14
10-10	TIMING_CFG_1 Field Descriptions	10-16
10-11	TIMING_CFG_2 Field Descriptions	10-18
10-12	DDR_SDRAM_CFG Field Descriptions.....	10-20
10-13	DDR_SDRAM_CFG_2 Field Descriptions.....	10-22
10-14	DDR_SDRAM_MODE Field Descriptions.....	10-24
10-15	DDR_SDRAM_MODE_2 Field Descriptions.....	10-24
10-16	DDR_SDRAM_MD_CNTL Field Descriptions.....	10-25
10-17	Settings of DDR_SDRAM_MD_CNTL Fields	10-26
10-18	DDR_SDRAM_INTERVAL Field Descriptions	10-27
10-19	DDR_DATA_INIT Field Descriptions	10-28
10-20	DDR_SDRAM_CLK_CNTL Field Descriptions	10-28
10-21	DDR_INIT_ADDR Field Descriptions	10-29
10-22	DDR_IP_REV1 Field Descriptions	10-29
10-23	DDR_IP_REV2 Field Descriptions	10-30
10-24	10-30
10-25	DATA_ERR_INJECT_LO Field Descriptions	10-31
10-26	ERR_INJECT Field Descriptions	10-31
10-27	CAPTURE_DATA_HI Field Descriptions	10-32
10-28	CAPTURE_DATA_LO Field Descriptions	10-32
10-29	CAPTURE_ECC Field Descriptions	10-33

Tables

Table Number	Title	Page Number
10-30	ERR_DETECT Field Descriptions	10-33
10-31	ERR_DISABLE Field Descriptions.....	10-34
10-32	ERR_INT_EN Field Descriptions	10-35
10-33	CAPTURE_ATTRIBUTES Field Descriptions	10-36
10-34	CAPTURE_ADDRESS Field Descriptions.....	10-37
10-35	ERR_SBE Field Descriptions	10-37
10-36	Byte Lane to Data Relationship	10-42
10-37	Supported DDR2 SDRAM Device Configurations	10-43
10-38	DDR2 Address Multiplexing for 32-Bit Data Bus with Interleaving and Partial Array Self Refresh Disabled	10-43
10-39	DDR2 Address Multiplexing for 16-Bit Data Bus (with Interleaving Disabled)	10-44
10-40	Example of Address Multiplexing for -Bit Data Bus Interleaving between Two Banks with Partial Array Self Refresh Disabled.....	10-45
10-41	DDR SDRAM Command Table.....	10-46
10-42	DDR SDRAM Interface Timing Intervals	10-47
10-43	DDR SDRAM Power-Saving Modes Refresh Configuration.....	10-55
10-44	Memory Controller–Data Beat Ordering	10-57
10-45	DDR SDRAM ECC Syndrome Encoding	10-58
10-46	DDR SDRAM ECC Syndrome Encoding (Check Bits)	10-59
10-47	Memory Controller Errors	10-60
10-48	Memory Interface Configuration Register Initialization Parameters.....	10-61
11-1	Signal Properties—Summary.....	11-4
11-2	Enhanced Local Bus Controller Detailed Signal Descriptions	11-5
11-3	Enhanced Local Bus Controller Registers	11-8
11-4	BR _n Field Descriptions	11-10
11-5	Reset value of OR0 Register	11-11
11-6	Memory Bank Sizes in Relation to Address Mask	11-12
11-7	OR _n —GPCM Field Descriptions	11-13
11-8	OR _n —FCM Field Descriptions	11-16
11-9	OR _n —UPM Field Descriptions	11-18
11-10	MAR Field Descriptions	11-19
11-11	M _x MR Field Descriptions.....	11-20
11-12	MRTPR Field Descriptions	11-22
11-13	MDR Field Description.....	11-23
11-14	LSOR Field Description.....	11-24
11-15	LURT Field Descriptions	11-25
11-16	LTESR Field Descriptions	11-26
11-17	LTEDR Field Descriptions.....	11-27
11-18	LTEIR Field Descriptions	11-28
11-19	LTEATR Field Descriptions.....	11-29
11-20	LTEAR Field Descriptions.....	11-30

Tables

Table Number	Title	Page Number
11-21	LBCR Field Descriptions.....	11-31
11-22	LCRR Field Descriptions.....	11-32
11-23	FMR Field Descriptions.....	11-33
11-24	FIR Field Descriptions.....	11-35
11-25	FCR Field Descriptions.....	11-36
11-26	FBAR Field Descriptions.....	11-36
11-27	FPAR Field Descriptions, Small Page Device (ORx[PGS] = 0).....	11-37
11-28	FPAR Field Descriptions, Large Page Device (ORx[PGS] = 1).....	11-37
11-29	FBCR Field Descriptions.....	11-38
11-30	GPCM Read Control Signal Timing.....	11-45
11-31	GPCM Write Control Signal Timing.....	11-46
11-32	Boot Bank Field Values after Reset for GPCM as Boot Controller.....	11-55
11-33	FCM Chip-Select to First Command Timing.....	11-64
11-34	FCM Command, Address, and Write Data Timing Parameters.....	11-65
11-35	FCM Read Data Timing Parameters.....	11-68
11-36	Boot Bank Field Values after Reset for FCM as Boot Controller.....	11-69
11-37	UPM Routines Start Addresses.....	11-72
11-38	RAM Word Field Descriptions.....	11-78
11-39	MxMR Loop Field Use.....	11-83
11-40	UPM Address Multiplexing.....	11-84
11-41	Data Bus Drive Requirements For Read Cycles.....	11-91
11-42	FCM Register Settings for Soft Reset (ORn[PGS] = 1).....	11-92
11-43	FCM Register Settings for Status Read (ORn[PGS] = 1).....	11-92
11-44	FCM Register Settings for ID Read (ORn[PGS] = 1).....	11-93
11-45	FCM Register Settings for Page Read (ORn[PGS] = 1).....	11-93
11-46	FCM Register Settings for Block Erase (ORn[PGS] = 1).....	11-94
11-47	FCM Register Settings for Page Program (ORn[PGS] = 1).....	11-95
12-1	Signal Properties.....	12-5
12-2	eSDHC Memory Map.....	12-6
12-3	DSADDR Field Descriptions.....	12-7
12-4	BLKATTR Field Descriptions.....	12-8
12-5	CMDARG Field Descriptions.....	12-9
12-6	XFERTYP Field Descriptions.....	12-9
12-7	Determination of Transfer Type.....	12-11
12-8	Relation Between Parameters and Name of Response Type.....	12-12
12-9	Response Bit Definition for Each Response Type.....	12-13
12-10	DATPORT Field Descriptions.....	12-14
12-11	PRSTAT Field Descriptions.....	12-15
12-12	PROCTL Field Descriptions.....	12-19
12-13	SYSCTL Field Descriptions.....	12-22
12-14	IRQSTAT Field Descriptions.....	12-25

Tables

Table Number	Title	Page Number
12-15	Relation Between Command Timeout Error and Command Complete Status	12-28
12-16	Relation Between Data Timeout Error and Transfer Complete Status	12-28
12-17	Relation Between Command CRC Error and Command Timeout Error	12-29
12-18	IRQSTATEN Field Descriptions	12-29
12-19	IRQSIGEN Field Descriptions	12-32
12-20	AUTO12ERR Field Descriptions	12-34
12-21	Relationship Between Command CRC Error and Command Timeout Error for Auto CMD12	12-34
12-22	HOSTCAPBLT Field Descriptions	12-36
12-23	WML Field Descriptions	12-37
12-24	FEVT Field Descriptions	12-38
12-25	HOSTVER Field Descriptions	12-39
12-26	Commands for MMC/SD/SDIO	12-61
12-27	EXT_CSD Access Modes	12-65
13-1	DMAC Register Summary	13-3
13-2	DMA Control Register (DMACR) Field Descriptions	13-4
13-3	DMAES Field Descriptions	13-6
13-4	DMAERQ Field Descriptions	13-8
13-5	DMAEEI Field Descriptions	13-9
13-6	DMASEEI Field Descriptions	13-10
13-7	DMACEEI Field Descriptions	13-10
13-8	DMACINT Field Descriptions	13-11
13-9	DMACERR Field Descriptions	13-12
13-10	DMASSRT Field Descriptions	13-12
13-11	DMACDNE Field Descriptions	13-13
13-12	DMAINT Field Descriptions	13-14
13-13	DMAERR Field Descriptions	13-15
13-14	DMAGPOR Field Descriptions	13-16
13-15	DCHPRI _n Field Descriptions	13-17
13-16	TCD 32-Bit Memory Structure	13-18
13-17	TCD Word 0 (TCD _n .saddr) Field Description	13-18
13-18	TCD Word 1 (TCD. _{smod, ssize, dmod, dsize, soff}) Field Descriptions	13-19
13-19	TCD Word 2 (TCD.nbytes) Field Description	13-20
13-20	TCD Word 3 (TCD.slast) Field Descriptions	13-21
13-21	TCD Word 4 (TCD.daddr) Field Description	13-21
13-22	TCD Word 5 (TCD. _{citer, doff}) Field Descriptions	13-22
13-23	TCD Word 6 (TCD.dlast_sga) Field Descriptions	13-23
13-24	TCD Word 7 (TCD. _{biter, control/status}) Field Descriptions	13-23
14-1	Module Memory Map	14-2
14-2	OMISR Field Descriptions	14-4
14-3	OMIMR Field Descriptions	14-5

Tables

Table Number	Title	Page Number
14-4	IMR0 and IMR1 Field Descriptions	14-5
14-5	OMR0 and OMR1 Field Descriptions	14-6
14-6	ODR Field Descriptions	14-6
14-7	IDR Field Descriptions	14-7
14-8	IMISR Field Descriptions	14-8
14-9	IMIMR Field Descriptions	14-8
14-10	DMAMR _n Field Descriptions	14-9
14-11	DMASR _n Field Descriptions	14-12
14-12	DMACDAR _n Field Descriptions	14-12
14-13	DMASAR _n Field Descriptions	14-13
14-14	DMASAR _n Field Descriptions	14-13
14-15	DMABCR _n Field Descriptions	14-14
14-16	DMANDAR _n Field Descriptions	14-14
14-17	DMA Segment Descriptor Fields	14-18
15-1	FlexCAN Signals	15-5
15-2	FlexCAN Memory Map	15-6
15-3	Message Buffer MB0 Memory Mapping	15-7
15-4	Message Buffer Description	15-7
15-5	Message Buffer Code for Rx buffers	15-9
15-6	Message Buffer Code for Tx buffers	15-9
15-7	ID Table (0–7) Field Description	15-11
15-8	Module Configuration Register Description	15-13
15-9	Control Register Description	15-16
15-10	Free Running Timer (TIMER) Register Description	15-19
15-11	Rx Global Mask Register Description	15-19
15-12	Error Counter Register Description	15-21
15-13	Error and Status Register Description	15-22
15-14	Interrupt Masks 2 Register (IMASK2) Description	15-25
15-15	Interrupt Masks 1 Register (IMASK1) Description	15-25
15-16	Interrupt Flags 2 Register (IFLAG2) Description	15-26
15-17	Interrupt Flags 1 Register (IFLAG2) Description	15-27
15-18	Interrupt Flags 2 Register (IFLAG2) Description	15-28
15-19	Time Segment Syntax	15-39
15-20	CAN Standard Compliant Bit Time Segment Settings	15-39
15-21	Minimum Ratio Between Peripheral Clock Frequency and CAN Bit Rate	15-40
16-1	ULPI Signal Descriptions	16-3
16-2	USB Interface Memory Map	16-4
16-3	CAPLENGTH Register Field Descriptions	16-6
16-4	HCIVERSION Register Field Descriptions	16-7
16-5	HCSPARAMS Register Field Descriptions	16-7
16-6	HCCPARAMS Register Field Descriptions	16-8

Tables

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

Tables

Table Number	Title	Page Number
16-48	siTD Transfer Status and Control.....	16-53
16-49	siTD Buffer Pointer Page 0 (Plus)	16-54
16-50	siTD Buffer Pointer Page 1 (Plus)	16-54
16-51	siTD Back Link Pointer	16-54
16-52	qTD Next Element Transfer Pointer (DWord 0).....	16-56
16-53	qTD Alternate Next Element Transfer Pointer (DWord 1)	16-56
16-54	qTD Token (DWord 2)	16-57
16-55	qTD Buffer Pointer	16-59
16-56	Queue Head DWord 0	16-60
16-57	Endpoint Characteristics: Queue Head DWord 1.....	16-61
16-58	Endpoint Capabilities: Queue Head DWord 2	16-62
16-59	Current qTD Link Pointer	16-63
16-60	Host-Controller Rules for Bits in Overlay (DWords 5, 6, 8, and 9)	16-64
16-61	FSTN Normal Path Pointer	16-65
16-62	FSTN Back Path Link Pointer	16-65
16-63	Behavior During Wake-Up Events.....	16-69
16-64	Operation of FRINDEX and SOFV (SOF Value Register).....	16-73
16-65	Example Periodic Reference Patterns for Interrupt Transfers	16-86
16-66	Ping Control State Transition Table	16-87
16-67	Interrupt IN/OUT Do Complete Split State Execution Criteria.....	16-101
16-68	Initial Conditions for OUT siTD TP and T-Count Fields	16-110
16-69	Transaction Position (TP)/Transaction Count (T-Count) Transition Table.....	16-110
16-70	Summary siTD Split Transaction State.....	16-114
16-71	Example Case 2a—Software Scheduling siTDs for an IN Endpoint.....	16-115
16-72	Summary of Transaction Errors	16-118
16-73	Summary Behavior on Host System Errors	16-121
16-74	Endpoint Capabilities/Characteristics	16-123
16-75	Current dTD Pointer.....	16-124
16-76	Multiple Mode Control	16-124
16-77	Next dTD Pointer	16-125
16-78	dTD Token	16-125
16-79	Buffer Pointer Page 0	16-126
16-80	Buffer Pointer Page 1	16-126
16-81	Buffer Pointer Pages 2–4	16-126
16-82	Device Controller State Information Bits	16-129
16-83	Device Controller Endpoint Initialization.....	16-132
16-84	Device Controller Stall Response Matrix	16-133
16-85	Variable Length Transfer Protocol Example (ZLT = 0).....	16-135
16-86	Variable Length Transfer Protocol Example (ZLT = 1).....	16-135
16-87	Interrupt/Bulk Endpoint Bus Response Matrix.....	16-136
16-88	Control Endpoint Bus Response Matrix	16-138

Tables

Table Number	Title	Page Number
16-89	Isochronous Endpoint Bus Response Matrix	16-141
16-90	Device Error Matrix	16-146
16-91	Error Descriptions	16-146
16-92	Interrupt Handling Order	16-146
16-93	Low Frequency Interrupt Events.....	16-147
16-94	Error Interrupt Events	16-147
16-95	Functional Differences Between EHCI and EHCI with Embedded TT	16-149
16-96	Emulated Handshakes	16-150
17-1	I ² C Interface Signal Descriptions	17-3
17-2	I ² C Interface Signals—Detailed Signal Descriptions	17-4
17-3	I ² C Memory Map	17-4
17-4	I2CnADR Field Descriptions.....	17-5
17-5	I2Cn FDR Field Descriptions	17-6
17-6	I2CnCR Field Descriptions	17-7
17-7	I2CnSR Field Descriptions	17-8
17-8	I2CnDR Field Descriptions.....	17-9
17-9	I2CnDFSRR 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 Descriptions	18-6
18-5	UTHR Field Descriptions	18-7
18-6	UDMB Field Descriptions	18-7
18-7	UDLB Field Descriptions	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	UAFR Field Descriptions.....	18-12
18-14	ULCR Field Descriptions.....	18-13
18-15	Parity Selection Using ULCR[PEN], ULCR[SP], and ULCR[EPS]	18-14
18-16	UMCR Field Descriptions	18-14
18-17	ULSR Field Descriptions	18-15
18-18	UMSR Field Descriptions	18-16
18-19	USCR Field Descriptions.....	18-17
18-20	UDSR Field Descriptions.....	18-17
18-21	UDSR[TXRDY] Set Conditions	18-18
18-22	UDSR[TXRDY] Cleared Conditions.....	18-18
18-23	UDSR[RXRDY] Set Conditions.....	18-18
18-24	UDSR[RXRDY] Cleared.....	18-18

Tables

Table Number	Title	Page Number
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 Field Descriptions.....	19-14
19-9	SPI Receive Data Hold Field Descriptions	19-15
20-1	JTAG Test Signals Summary	20-2
20-2	JTAG Test—Detailed Signal Descriptions.....	20-2
21-1	IPIC External Signals—Detailed Signal Descriptions.....	21-2
21-2	GPIO Register Address Map.....	21-2
21-3	GPnDIR Bit Settings	21-3
21-4	GPnODR Bit Settings	21-4
21-5	GPnDAT Bit Settings	21-4
21-6	GPnIER Bit Settings	21-5
21-7	GPnIMR Bit Settings	21-5
21-8	GPnICR Bit Settings	21-6
22-1	Sequencer Memory Map.....	22-2
22-2	POTAR _n Field Descriptions	22-3
22-3	POBAR _n Field Descriptions.....	22-4
22-4	POCMR _n Field Descriptions	22-4
22-5	PMCR Field Descriptions	22-5
22-6	DTCR Field Descriptions.....	22-6
23-1	PCI Controller Modes	23-3
23-2	Signal Properties	23-4
23-3	PCI Interface Signals—Detailed Signal Descriptions	23-5
23-4	PCI Configuration Access Registers.....	23-11
23-5	PCI Memory-Mapped Registers	23-11
23-6	PCI_CONFIG_ADDRESS Field Descriptions.....	23-13
23-7	PCI_CONFIG_DATA Field Descriptions.....	23-15
23-8	PCI_ESR Field Descriptions.....	23-16
23-9	PCI_ECDR Field Descriptions	23-17
23-10	PCI_EER Field Descriptions	23-17
23-11	PCI_EATCR Field Descriptions	23-18
23-12	PCI_EACR Field Description	23-19
23-13	PCI_EEACR Field Description	23-20
23-14	PCI_EDCR Field Description.....	23-20
23-15	PCI_GCR Field Descriptions.....	23-21
23-16	PCI_ECR Field Descriptions	23-21

Tables

Table Number	Title	Page Number
23-17	PCI_GSR Field Descriptions	23-22
23-18	PITAR _n Field Descriptions	23-23
23-19	PIBAR _n Field Descriptions	23-23
23-20	PIEBAR _n Field Descriptions	23-24
23-21	PIWAR _n Field Descriptions	23-24
23-22	PCI Configuration Space Registers.....	23-25
23-23	Vendor ID Configuration Register Field Descriptions.....	23-27
23-24	Device ID Configuration Register Field Descriptions	23-27
23-25	PCI Command Configuration Register Field Descriptions.....	23-27
23-26	PCI Status Configuration Register Field Descriptions.....	23-28
23-27	Revision ID Configuration Register Field Descriptions	23-29
23-28	Standard Programming Interface Configuration Register Field Descriptions	23-30
23-29	Subclass Code Configuration Register Field Descriptions	23-30
23-30	Class Code Configuration Register Field Descriptions	23-31
23-31	Cache Line Size Configuration Register Field Descriptions	23-31
23-32	Latency Timer Configuration Register Field Descriptions	23-32
23-33	PIMMR Base Address Configuration Register Field Descriptions	23-33
23-34	GPL Base Address Register 0 Field Descriptions	23-34
23-35	GPL Base Address Register 1,2 Field Descriptions	23-34
23-36	GPL Extended Base Address Registers 1–2 Field Descriptions	23-35
23-37	Subsystem Vendor ID Configuration Register Field Descriptions	23-35
23-38	Subsystem Device ID Configuration Register Field Descriptions.....	23-36
23-39	Interrupt Line Configuration Register Field Descriptions	23-37
23-40	PCI Function Configuration Register Field Descriptions	23-38
23-41	PCI Arbiter Control Register (PCIACR) Field Descriptions.....	23-39
23-42	Hot Swap Register Block Field Descriptions	23-40
23-43	PCI Command Definitions.....	23-43
24-1	QEIWRM Chapters and 8309 Implementation.....	24-3
24-2	SDSR Field Descriptions	24-6
24-3	SDMR Field Descriptions	24-7
24-4	SDTR Field Descriptions	24-10
24-5	SDHY _x Field Descriptions	24-11
24-6	SDTAX Field Descriptions	24-12
24-7	SDTM _x Field Descriptions	24-12
24-8	SDEBCR Field Descriptions.....	24-13
24-9	Interrupt Source Priority Levels.....	24-14
24-10	Mapping of FCCs and SCCs to UCCs for 82xx/85xx Compatibility	24-18
24-11	Encoding the Interrupt Vector	24-19
24-12	CICR Field Descriptions	24-22
24-13	CICNR Bit Settings.....	24-23
24-14	CRICR Bit Settings	24-25

Tables

Table Number	Title	Page Number
24-15	CIPWCC Field Descriptions	24-26
24-16	CIPXCC Field Descriptions	24-27
24-17	CIPYCC Field Descriptions	24-27
24-18	CIPZCC Field Descriptions	24-28
24-19	CIPRTA Field Descriptions.....	24-29
24-20	CIPRTB Field Descriptions	24-30
24-21	Default Parameter RAM Base Addresses	24-35
24-22	QUICC Engine Parameter RAM Base Addresses (Suggested Value for User Configuration)	24-35
24-23	IReady Field Descriptions.....	24-36
24-24	CEVTxxER Field Descriptions.....	24-39
24-25	CEVTxxMR Field Descriptions.....	24-39
24-26	CEVTPER Field Descriptions	24-40
24-27	CEVTPMR Field Descriptions	24-40

Tables

**Table
Number**

Title

**Page
Number**

About This Book

This reference manual defines the functionality of the MPC8309. The MPC8309 is a cost-effective, highly integrated communications processor that addresses the requirements of several networking applications, including residential gateways, modem/routers, industrial control, and test and measurement applications.

The MPC8309 extends current PowerQUICC offerings, adding higher CPU performance, additional functionality, and faster interfaces, while addressing the requirements related to time-to-market, price, power consumption, and board real estate.

Organization

The major parts of this reference manual are briefly described as follows:

- [Chapter 1, “Overview,”](#) provides a high-level description of features and functionality of the MPC8309. 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 memory map of the device. An overview of the local address map is provided. Next, a complete listing of all memory-mapped registers is provided, with cross references to the sections detailing descriptions of each.
- [Chapter 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. Also, these signals are listed by 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 Boot,”](#) describes the overall initialization of the MPC8309.
- [Chapter 6, “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 7, “Arbiter and Bus Monitor,”](#) provides an overview of the arbiter in the device. Also, it describes the configuration, control, and status registers of the arbiter.
- [Chapter 8, “e300 Processor Core Overview,”](#) provides an overview of the basic functionality of the processor core and briefly describes how the functional units interact.
- [Chapter 9, “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 10, “DDR Memory Controller,”](#) describes the DDR2 memory controller of the device. This fully programmable controller supports most DDR memories available today, including both buffered and unbuffered devices. Dynamic power management and auto-precharge modes simplify memory system design.
- [Chapter 11, “Enhanced Local Bus Controller,”](#) describes the enhanced local bus controller (eLBC) of the device. It describes the external signals and the memory-mapped registers as well as a functional description of the general-purpose chip-select machine (GPCM), Flash control machine (FCM), and user-programmable machines (UPMs) of the eLBC. Also, it includes an initialization and applications information section with many specific examples of its use.
- [Chapter 12, “Enhanced Secure Digital Host Controller,”](#) describes the enhanced SD Host Controller, which provides an interface between the host system and SD/MMC/SDIO cards. It provides a functional description of the major system blocks and includes command information for the host.
- [Chapter 13, “DMA Engine 1,”](#) describes a second-generation platform module capable of performing complex data transfers with minimal intervention from a host processor through 16 programmable channels. The hardware micro-architecture includes a DMA engine, which performs source and destination address calculations, and the actual data movement operations, along with a local memory containing the transfer control descriptors (TCD) for the channels.
- [Chapter 14, “DMA Engine 2,”](#) describes the DMA Engine 2, which supports communication between two processors on different buses.
- [Chapter 15, “FlexCAN,”](#) describes the FlexCAN module, which is a communication controller implementing the CAN protocol according to the CAN 2.0B protocol specification.
- [Chapter 16, “Universal Serial Bus Interface,”](#) describes the universal serial bus (USB) interface. The USB DR module is a USB 2.0-compliant serial interface engine for implementing a USB interface. The DR module supports the required signaling for UTMI low pin count interface (ULPI) transceivers (PHYS). An external PHY would be used to interface to ULPI.
- [Chapter 17, “I²C Interfaces,”](#) describes the inter-IC (IIC or I²C) bus controllers of the device. These synchronous, serial, bidirectional, multiple-master buses allow two-wire connection of devices, such as microcontrollers, EEPROMs, real-time clock devices, A/D converters, and LCDs. The device powers up in boot sequencer mode, which allows the I²C controllers to initialize configuration registers.
- [Chapter 18, “DUART,”](#) describes the (dual) universal asynchronous receiver/transmitters (UARTs) which feature a PC16552D-compatible programming model. These independent UARTs are provided specifically to support system debugging.
- [Chapter 19, “Serial Peripheral Interface,”](#) describes the MPC8309 serial peripheral interface (SPI) that allows the exchange of data between the MPC8309 and MPC83xx family of devices. The SPI can also be used to communicate with peripheral devices such as EEPROMs, real-time clocks, A/D converters, and ISDN devices.
- [Chapter 20, “JTAG/Testing Support,”](#) describes the joint test action group (JTAG) interface of the MPC8309 to facilitate boundary-scan testing. The JTAG interface complies to the IEEE 1149.1™ boundary-scan specification.

- [Chapter 21, “General Purpose I/O \(GPIO\),”](#) describes the general purpose I/O (GPIO) module in the MPC8309, including a definition of the external signals and functions they serve. Additionally, interrupt capabilities, pin description, and register settings are described.
- [Chapter 22, “Sequencer,”](#) describes the I/O sequencer, which switches transactions among its ports, using a buffer pool to minimize blocking.
- [Chapter 23, “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 24, “QUICC Engine Block on the MPC8309,”](#) serves as both a general overview of the QUICC Engine block and a guide to the specific implementation of the QUICC Engine block on the MPC8309.
- [Appendix A, “Complete List of Configuration, Control, and Status Registers,”](#) lists all the registers used with MPC8309.
- [Appendix B, “Revision History,”](#) lists the major differences between revisions of the *MPC8309 PowerQUICC II Pro Integrated Communications Processor Reference Manual*.

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:

- *e300 Core Reference Manual*—This book provides a more detailed description of the e300 core.
- *MPC8309 PowerQUICC II Pro Processor Device Errata*—This document details all known silicon errata for the Freescale IPs in the MPC8309.
- *MPC8309 PowerQUICC II Pro Integrated Communications Processor Hardware Specification*—This document provides an overview of the MPC8309 features, its hardware specifications, including a block diagram showing the major functional components. Hardware

specifications provide specific data regarding bus timing, signal behavior, and AC, DC, and thermal characteristics, as well as other design considerations.

For more information on other device documentation, refer to <http://www.freescale.com>.

Conventions

This document uses the following notational conventions:

cleared/set	When a bit takes the value zero, it is said to be cleared; when it takes a value of one, it is said to be set.				
mnemonics	Instruction mnemonics are shown in lowercase bold				
<i>italics</i>	Italics indicate variable command parameters, for example, bcctr <i>x</i> Book titles in text are set in italics Internal signals are set in lowercase italics, for example, <u><i>core_int</i></u>				
0x0	Prefix to denote hexadecimal number				
0b0	Prefix to denote binary number				
rA, rB	Instruction syntax used to identify a source GPR				
rD	Instruction syntax used to identify a destination GPR				
REG[FIELD]	Abbreviations for registers are shown in uppercase text. Specific bits, fields, or ranges appear in brackets. For example, MSR[LE] refers to the little-endian mode enable bit in the machine state register.				
x	In some contexts, such as signal encodings, an unitalicized x indicates a don't care				
<i>x</i>	An italicized <i>x</i> indicates an alphanumeric variable				
<i>n</i>	An italicized <i>n</i> indicates a numeric variable				
¬	NOT logical operator				
&	AND logical operator				
	OR logical operator				
	Concatenation, for example TCR[WP] TCR[WPEXT]				
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td style="width: 20px; height: 15px;">—</td></tr> </table>	—	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.			
—					
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td style="width: 20px; height: 15px;">R</td><td style="width: 20px; height: 15px;">0</td></tr> <tr><td style="width: 20px; height: 15px;">W</td><td style="width: 20px; height: 15px;"></td></tr> </table>	R	0	W		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.
R	0				
W					
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td style="width: 20px; height: 15px;">R</td><td style="width: 20px; height: 15px;">FIELDNAME</td></tr> <tr><td style="width: 20px; height: 15px;">W</td><td style="width: 20px; height: 15px;"></td></tr> </table>	R	FIELDNAME	W		Indicates a read-only bit field in a memory-mapped register.
R	FIELDNAME				
W					
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td style="width: 20px; height: 15px;">R</td><td style="width: 20px; height: 15px;"></td></tr> <tr><td style="width: 20px; height: 15px;">W</td><td style="width: 20px; height: 15px;">FIELDNAME</td></tr> </table>	R		W	FIELDNAME	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.
R					
W	FIELDNAME				

Signal Conventions

- OVERBAR An overbar indicates that a signal is active-low.
- lowercase_italics* Lowercase italics is used to indicate internal signals.
- lowercase_plaintext Lowercase plain text is used to indicate signals that are used for configuration.

Acronyms and Abbreviations

Table i contains acronyms and abbreviations used in this document.

Table i. Acronyms and Abbreviated Terms

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

Table i. Acronyms and Abbreviated Terms (continued)

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

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
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
SCL	Serial clock
SDA	Serial data
SFD	Start frame delimiter
SI	Serial interface
SPI	Serial peripheral interface
SPR	Special-purpose register
SRAM	Static random access memory
TAP	Test access port
TBI	Ten-bit interface
TLB	Translation lookaside buffer
TSEC	Three-speed Ethernet controller
Tx	Transmit
TxBD	Transmit buffer descriptor
UART	Universal asynchronous receiver/transmitter
UPM	User-programmable machine
UTP	Unshielded twisted pair
WDT	Watchdog timer
ZBT	Zero bus turnaround

Chapter 1

Overview

This chapter provides an overview of the MPC8309 PowerQUICC II Pro processor features, including a block diagram showing the major functional components. The MPC8309 is a cost-effective, highly integrated communication processors. The device extends the PowerQUICC family, adding higher CPU performance, additional functionality, and faster interfaces while addressing the requirements related to time-to-market, price, power consumption, and package size.

1.1 MPC8309 PowerQUICC II Pro Processor Overview

Figure 1-1 shows the major functional units within the MPC8309. The e300c3 core in the MPC8309, with its 16 Kbytes of instruction and 16 Kbytes of data cache, implements the PowerPC user instruction set architecture and provides hardware and software debugging support. Another is the QUICC Engine, a communications complex based on QUICC Engine, which provides termination and switching between a wide range of protocols including Ethernet, IEEE 1588, TDM, and HDLC. Other major features include a DDR2 SDRAM memory controller with 8 bit ECC, an enhanced local bus (eLBC), an enhanced secure digital host controller (eSDHC), a 32-bit PCI-2.3 controller, an integrated programmable interrupt controller (IPIC), a general purpose DMA controller, four FlexCAN interfaces with maximum 64 message buffers each, two I²C controllers, two DUARTs, GPIOs, USB, one general purpose timer block, and an SPI controller.

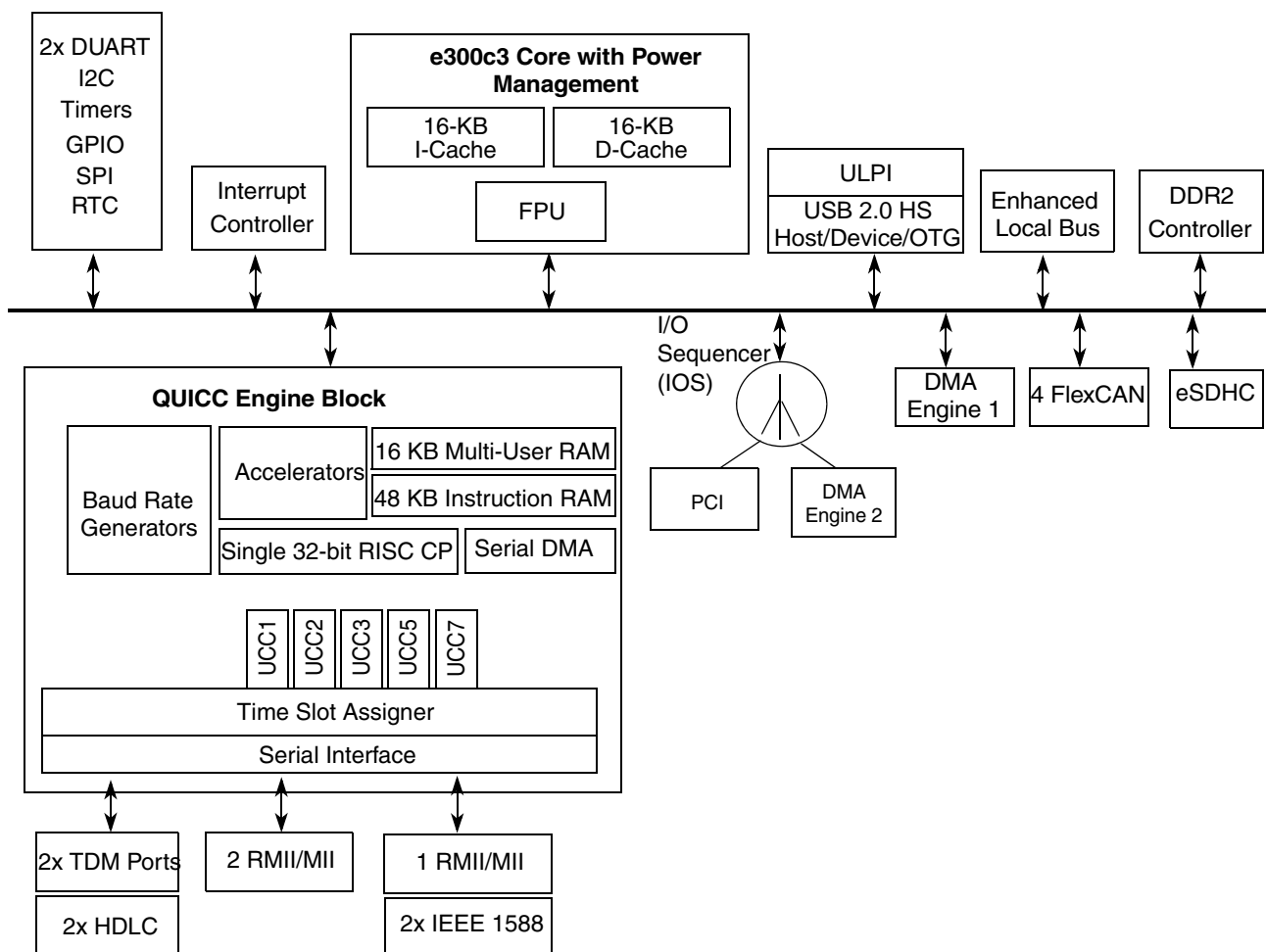


Figure 1-1. MPC8309 Block Diagram

1.2 Features

The major features of this device are as follows:

- e300c3 Power Architecture® processor core
 - Enhanced version of the MPC603e core
 - High-performance, superscalar processor core with a four-stage pipeline and low interrupt latency times
 - Floating-point, dual integer units, load/store, system register, and branch processing units
 - 16-Kbyte instruction cache and 16-Kbyte data cache with lockable capabilities
 - Capable of completing two MACs every 3 cycles
 - Dynamic power management
 - Enhanced hardware program debug features

- Software-compatible with Freescale processor families implementing Power Architecture technology
- Separate PLL that is clocked by the system bus clock
- Performance monitor
- QUICC Engine block
 - 32-bit RISC controller for flexible support of the communications peripherals with the following features:
 - One clock per instruction
 - Separate PLL for operating frequency that is independent of system's bus and e300 core frequency for power and performance optimization
 - 32-bit instruction object code
 - Executes code from internal IRAM
 - 32-bit arithmetic logic unit (ALU) data path
 - Modular architecture allowing for easy functional enhancements
 - Slave bus for CPU access of registers and multiuser RAM space
 - 48 Kbytes of instruction RAM
 - 16 Kbytes of multiuser data RAM
 - Serial DMA channel for receive and transmit on all serial channels
 - Five unified communication controllers (UCCs) supporting the following protocols and interfaces:
 - 10/100 Mbps Ethernet/IEEE® Std. 802.3® through MII and RMII interfaces.
 - IEEE Std. 1588™ support
 - HDLC/Transparent (bit rate up to QUICC Engine operating frequency / 8)
 - HDLC Bus (bit rate up to 10 Mbps)
 - Asynchronous HDLC (bit rate up to 2 Mbps)
 - Multi-channel controller on UCC (UMCC) controller functionality, which can emulate time-division serial channels using a single unified communication controller (UCC) and a time-division multiplexed (TDM) physical interface

For more information on QUICC Engine sub-modules, see *QUICC Engine™ Block Reference Manual with Protocol Interworking*.

- DDR SDRAM memory controller
 - Programmable timing supporting DDR2 SDRAM
 - Integrated SDRAM clock generation
 - Supports 8 bit ECC
 - 16/32-bit data interface, up to 266-MHz data rate
 - 14 address lines
 - The following SDRAM configurations are supported:
 - Up to two physical banks (chip selects), each bank upto 512-Mbyte addressable space for 32 bit data interface

- 64-Mbit to 2-Gbit devices with x8/x16/x32 data ports (no direct x4 support)
- One 16-bit device or two 8-bit devices on a 16-bit bus, or two 16-bit devices or four 8-bit devices on a 32-bit bus
- Support for up to 16 simultaneous open pages for DDR2
- Two clock pairs to support up to 4 DRAM devices
- Supports auto refresh
- On-the-fly power management using CKE
- PCI interface
 - Designed to comply with *PCI Local Bus Specification, Revision 2.3*
 - 32-bit PCI interface operating at up to 66 MHz
 - PCI 3.3-V compatible
 - Not 5-V compatible
 - Support for host and agent modes
 - 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 three masters on PCI
 - Arbiter support for two-level priority request/grant signal pairs
 - 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
 - Mapping from an external 32-/64-bit address space to the internal 32-bit local space
 - Support for dual address cycle (DAC) (as a target only)
 - Internal configuration registers accessible from PCI
 - Selectable snooping for inbound transactions
 - Four outbound translation address windows
 - Support for mapping 32-bit internal local memory space to an external 32-bit PCI address space and translating that address within the PCI space
 - Four inbound translation address windows corresponding to defined PCI BARs
 - The first BAR is 32-bits and dedicated to on-chip register access
 - The second BAR is 32-bits for general use
 - The remaining two BARs may be 32- or 64-bits and are also for general use
- Enhanced local bus controller (eLBC)
 - Multiplexed 26-bit address and 8-/16-bit data operating at up to 66 MHz
 - Eight chip selects supporting eight external slaves
 - Four chip selects dedicated
 - Four chip selects offered as multiplexed option

- Supports boot from parallel NOR Flash and parallel NAND Flash
- Supports programmable clock ratio dividers
- Up to eight-beat burst transfers
- 16- and 8-bit ports, separate $\overline{\text{LWE}}$ for each 8 bit
- Three protocol engines available on a per chip select basis:
 - General-purpose chip select machine (GPCM)
 - Three user programmable machines (UPMs)
 - NAND Flash control machine (FCM)
- Variable memory block sizes for FCM, GPCM, and UPM mode
- Default boot ROM chip select with configurable bus width (8 or 16)
- Provides two Write Enable signals to allow single byte write access to external 16-bit eLBC slave devices
- Integrated programmable interrupt controller (IPIC)
 - Functional and programming compatibility with the MPC8260 interrupt controller
 - Support for external and internal discrete interrupt sources
 - Programmable highest priority request
 - Six groups of interrupts with programmable priority
 - External and internal interrupts directed to host processor
 - Redirects interrupts to external $\overline{\text{PCI_INTA}}$ signal when in core disable mode
 - Unique vector number for each interrupt source
- Enhanced secure digital host controller (eSDHC)
 - Compatible with the *SD Host Controller Standard Specification Version 2.0* with test event register support
 - Compatible with the *MMC System Specification Version 4.2*
 - Compatible with the *SD Memory Card Specification Version 2.0* and supports the high capacity SD memory card
 - Compatible with the *SD Input/Output (SDIO) Card Specification, Version 2.0*
 - Designed to work with SD Memory, miniSD Memory, SDIO, miniSDIO, SD Combo, MMC, MMC*plus*, and RS-MMC cards
 - Card bus clock frequency up to 31.25 MHz
 - Supports 1-/4-bit SD and SDIO modes, 1-/4-bit modes
 - Up to 133 Mbps data transfer for SD/SDIO/MMC cards using 4 parallel data lines
 - Supports block sizes of 1 ~ 4096 bytes
- Universal serial bus (USB) dual-role controller
 - Designed to comply with *Universal Serial Bus Revision 2.0 Specification*
 - Supports operation as a stand-alone USB host controller
 - Supports operation as a stand-alone USB device

- Supports high-speed (480-Mbps), full-speed (12-Mbps), and low-speed (1.5-Mbps) operations. Low speed is only supported in host mode.
- FlexCAN module
 - Full implementation of the CAN protocol specification version 2.0B
 - Up to 64 flexible message buffers of zero to eight bytes data length
 - Powerful Rx FIFO ID filtering, capable of matching incoming IDs
 - Selectable backwards compatibility with previous FlexCAN module version
 - Programmable loop-back mode supporting self-test operation
 - Global network time, synchronized by a specific message
 - Independent of the transmission medium (an external transceiver is required)
 - Short latency time due to an arbitration scheme for high-priority messages
- Dual I²C interfaces
 - Two-wire interface
 - Multiple-master support
 - Master or slave I²C mode support
 - On-chip digital filtering rejects spikes on the bus
 - I²C 1 can be used as the boot sequencer
- DMA Engine 1
 - Support for the DMA engine with the following features:
 - Sixteen DMA channels
 - All data movement via dual-address transfers: read from source, write to destination
 - Transfer control descriptor (TCD) organized to support two-deep, nested transfer operations
 - Channel activation via one of two methods (for both the methods, one activation per execution of the minor loop is required):
 - Explicit software initiation
 - Initiation via a channel-to-channel linking mechanism for continuous transfers (independent channel linking at end of minor loop and/or major loop)
 - Support for fixed-priority and round-robin channel arbitration
 - Channel completion reported via optional interrupt requests
 - Support for scatter/gather DMA processing
- I/O sequencer
- Direct memory access (DMA) controller (DMA Engine 2)
 - Four independent fully programmable DMA channels
 - Concurrent execution across multiple channels with programmable bandwidth control
 - Misaligned transfer capability for source/destination address
 - Data chaining and direct mode
 - Interrupt on completed segment, error, and chain

- DUART
 - Supports two DUART blocks
 - Each DUART comprises a 2-wire interface (RxD, TxD).
 - A DUART, ie. two UART can be configured to become one UART with a 4-wire interface (RxD, TxD, RTS, CTS).
 - Programming model compatible with the original 16450 UART and the PC16550D
- Serial peripheral interface (SPI)
 - Master or slave support
- Power management controller (PMC)
 - Support core nap/sleep power management
 - Exits low power state and returns to full-on mode on
 - The core internal time base unit invokes a request to exit low power state
 - The power management controller detects that the system is not idle and there are outstanding transactions on the internal bus
- Parallel I/O
 - General-purpose I/O (GPIO)
 - 64 parallel I/O pins multiplexed on various chip interfaces
 - Interrupt capability
- System timers
 - Periodic interrupt timer
 - Software watchdog timer
 - Eight general-purpose timers
- Real time clock (RTC) module
 - Maintains a one-second count, unique over a period of thousand of years
 - Two possible clock sources:
 - External RTC clock (RTC_PIT_CLK)
 - CSB bus clock
- IEEE Std. 1149.1™ compliant JTAG boundary scan

1.3 MPC8309 Architecture Overview

The following sections describe the major functional units of this deviceMPC8309.

1.3.1 Power Architecture Core

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

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

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

The e300 core is a low-power implementation of the family of microprocessors that implements Power Architecture technology. The core implements the 32-bit portion of the architecture, 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 completes and retires as many as two instructions per clock cycle. Instructions can execute out of order for increased performance; however, the core makes completion appear sequential.

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

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

As an added feature to the e300 core, the device can lock the contents of three of the four ways in the instruction and data cache (or an entire cache). For example, this allows embedded applications to lock interrupt routines or other important (time-sensitive) instruction sequences into the instruction cache. It allows data to be locked into the data cache, which may be important to code that must have deterministic execution.

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

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

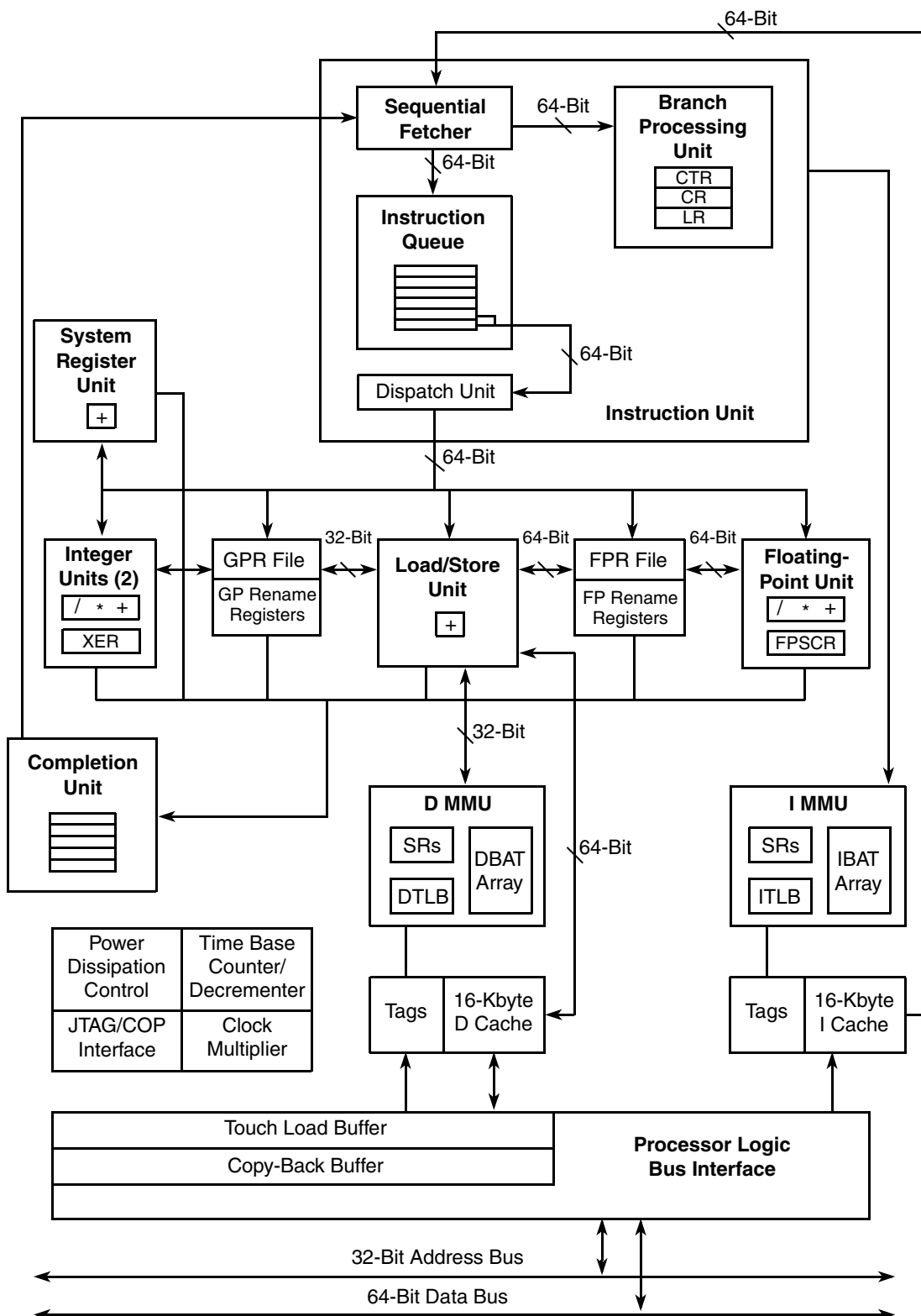


Figure 1-2. MPC8309 Integrated e300 Core Block Diagram

1.3.2 QUICC Engine Block

The QUICC Engine block is a versatile communications complex that integrates several communications peripheral controllers. It provides an on-chip system design that can be used as a building block for chip integration in a variety of applications, particularly in communications and networking systems.

The QUICC Engine block is the next generation of the Power QUICC II CPM and maintains a high level of compatibility with it. It contains the following communication peripherals:

- Five unified communication controllers (UCCs) with the following features:
 - 10/100 Mbps Ethernet/IEEE Std. 802.3 through MII and RMII interfaces
 - IEEE Std. 1588 support for MPC8309
 - Ethernet, HDLC/HDLC bus, and transparent protocols (also known as fast protocols).
 - Ethernet for the First Mile (IEEE 802.3ah 2BASE-TL and 10PASS-TS)
 - Async HDLC that are user compatible with the SCC of the CPM
 - The HDLC and transparent protocols are user compatible with the FCC of the CPM.
- Time slot assigner and serial interface (SI) for 2 TDMs and full duplex routing RAM of 512 entries.

The UCCs are similar to the PowerQUICC II peripherals: SCC (HDLC bus), and FCC (fast Ethernet, HDLC, and transparent).

Figure 1-3 shows the internal architecture and the interfaces provided by the QUICC Engine block. The QUICC Engine block contains five UCCs controlled by a single RISC engine. A common multiuser RAM is used to store parameters for RISC engine. The instruction RAM is used to run RISC code from the RAM.

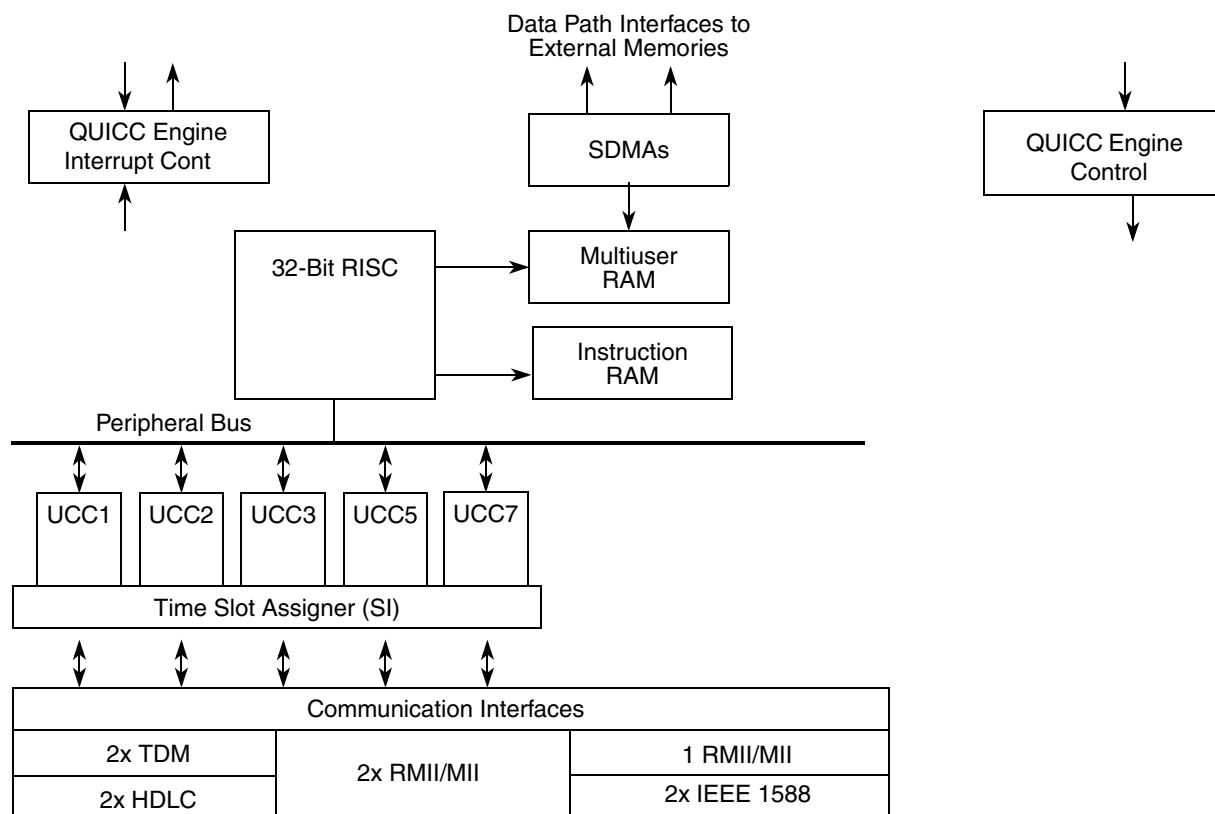


Figure 1-3. QUICC Engine Block Architectural Block Diagram

1.3.2.1 QUICC Engine Interfaces

The following sections describe the different QUICC Engine interfaces.

1.3.2.1.1 System Interface

The QUICC Engine block communicates with the CPU core in the following ways:

- Many parameters are exchanged through the multiuser RAM.
- The QUICC Engine block can execute special commands issued by the CPU. These commands should only be issued in special situations such as exceptions and error recovery.
- The QUICC Engine block generates interrupts through the system interface (SI) interrupt controller.
- The CPU can read/clear the QUICC Engine block status/event registers at any time.

1.3.2.1.2 QUICC Engine Communication Interfaces

The following are the total number of interfaces in the QUICC Engine block:

- One of the 3 RMII/MII interfaces is multiplexed with IEEE 1588 functions.
- 2 TDMs, each containing data, clocks, sync, and strobcs
- Some of the interfaces are multiplexed and cannot be used simultaneously.

1.3.2.2 Differences Between the QUICC Engine Block and the MPC82xx/85xx CPM

The following MPC82xx/MPC85xx features have been modified for the QUICC Engine block:

- SCC DPLL is not provided
- SCC 10 Base T (7-wire Ethernet) is no longer provided
- HDLC bus protocol programming model is FCC- instead of SCC-compatible
- Enhanced Ethernet controller which provides support for frame filtering based on the VLAN tag or any Ethernet type field and parsing of frame headers to perform table lookups
- General circuits interface (GCI) circuits through the serial interface are not supported.
- The instruction RAM is indirectly accessed.

1.3.2.3 Enhanced Features of the QUICC Engine Block Compared with the CPM

The following list highlights some significant improvements in the QUICC Engine block:

- Enhanced Ethernet features that provide for:
 - Frame filtering based on the MAC destination and source address, VLAN tag field and parsing of frame headers to perform table lookups.
 - IP support for IPv4 and IPv6 packets including TOS and header checksum processing.
 - UEC controller for 10/100 Mbps with support of VLAN
- IEEE Std. 1588 support
 - The IEEE Std. 1588 hardware assistance is supported on the RMII/MII 10/100 Mbps interfaces (full and half duplex)
 - Per packet timestamp tag for Rx
 - Programmable timestamp capture for Tx
 - Recognition of PTP packet – periodic pulse generation
 - Self-correcting precision timer with sub micro-second resolution
 - Phase aligned adjustable (divide by N) clock output
 - Two 64-bit alarm (future time) registers that hold the value of the future time for comparison
 - Maskable interrupts on alarm
 - Programmable polarity for both alarm (future time) output signals
 - Supports two timestamp trigger pins
- User can modify the peripheral's (UCC) parameter RAM base address in the multi-user RAM
- User programmable FIFO size for UCC fast protocols.
- The QUICC Engine operating frequency is independent of the SIU bus frequency, providing greater performance/power flexibility.
- Two independent time-stamp registers triggered by an external or internal clock

1.3.2.4 Software Migration from the MPC82xx/MPC85xx Family Devices

The QUICC Engine block was designed to minimize the changes required in run time code developed for the PowerQUICC II in order to ease the code porting from previous PowerQUICC II and CPM enabled devices (MPC82xx family, MPC85xx CPM enabled derivatives) to this device. Special attention was given to maintain compatibility with interrupts, events, status, interrupt event queues and data descriptors. However, significant changes have been made in the Ethernet controllers yielding the significant increase in performance when using those protocols.

Although some registers are new, many registers in the QUICC Engine block retain the previous mode, status, and event bits. The buffer descriptor method of transferring data from the QUICC Engine block to the CPU is maintained. The initialization code differs from that of the MPC82xx.

The UCC is a unification between the SCC and the FCC in the MPC85xx/MPC82xx families of devices. In HDLC, transparent, and Ethernet modes, the UCC programming model is compatible with the FCC.

For all other protocols (HDLC, transparent, Async HDLC, HDLC Bus, channelized HDLC/transparent), the QUICC Engine initialization is almost identical to the MPC82xx/MPC85xx.

1.3.2.5 Serial Protocol Table

Table 1-1 summarizes the available protocols for each serial port.

Table 1-1. QUICC Engine Protocols

Protocol	Controller	
	UCC	SI
Ethernet	√	—
HDLC	√	—
HDLC_BUS	√	—
Async HDLC	√	—
Transparent	√	—
Multi channel HDLC/Transparent TDM	√	√
Ethernet management (SMI)	√ (Opt)	—
IEEE 1588	√	—

1.3.2.6 QUICC Engine UCC Capabilities

Not all the UCCs in the QUICC Engine block can be configured for all of the protocols. [Table 1-2](#) lists the available protocols per UCC.

Table 1-2. UCC Protocol Enablement in the QUICC Engine Block

Protocol	Controller				
	UCC1	UCC2	UCC3	UCC5	UCC7
Ethernet	√	√	√	—	—
NMSI (HDLC/ Transparent/Async HDLC)	—	—	—	√	√
IEEE 1588	√	√	—	—	—

1.3.2.7 QUICC Engine Configurations

The QUICC Engine block offers configuration flexibility for specific applications. The previously-mentioned functions are all available, but not all of them can be used at the same time. The two physical factors that limit the functionality in any given system are performance and pinout.

Contact a Freescale FAE for more information on serial performance.

1.3.3 DDR2 Memory Controller

This fully programmable DDR2 SDRAM controller supports most JEDEC standard x8 or x16 DDR2 memories available today. The DDR memory controller include the following features:

- Support for DDR2 SDRAM
- 16- or 32-bit SDRAM data bus
- Programmable settings for meeting SDRAM timing parameters
- Many different SDRAM configurations supported
 - Support for two physical banks (chip selects)
 - Support for 64-Mbit to 2-Gbit devices with x8/x16/x32 data ports (no direct x4 support).
- Support for data mask signals and read-modify-write operations for sub-double word writes
- Supports 8-bit ECC
- Open page management (dedicated entry for each sub-bank)
- Support for error injection on ECC

1.3.4 PCI Bus Interface

The 32-bit PCI controller is compatible with the *PCI Local Bus Specification, Rev. 2.3*. The PCI interface can function as a host bridge interface. The PCI interface can optionally function as an agent device. The PCI controller supports 32-bit addressing and 32-bit data buses. As a host, 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 an agent, 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 controller includes the following distinctive features:

- Address stepping on configuration transactions
- Fast back-to-back transactions
- Data streaming
- Supports mapping from an external 32- or 64-bit address space to the internal 32-bit local space
- Supports dual address cycle (DAC) 64-bit addressing mode as target only
- When in host mode, the PCI controller supports external signal isolation, thus enabling power shut off to external devices

1.3.4.1 PCI Bus Arbitration Unit

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

- Supports three REQ/GNT signal pairs, thus supporting three external masters. The device PCI 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.

1.3.5 I/O Sequencer

The I/O sequencer switches transactions among the PCI, DMA Engine 2, and internal CSB ports, using a buffer pool to minimize blocking. The I/O sequencer includes the following features:

- Switches transactions among its ports
- Contains 8 cache-line (32 byte) buffers to allow streaming of PCI transactions
- Performs address translation on outbound PCI transactions

1.3.6 Enhanced Local Bus Controller (eLBC)

The main component of the enhanced local bus controller (eLBC) is its memory controller, which provides a seamless interface to many types of memory devices and peripherals. The memory controller is responsible for controlling eight memory banks shared by a NAND Flash control machine (FCM), a general-purpose chip-select machine (GPCM), and up to three user-programmable machines (UPMs). As such, it supports a minimal glue logic interface to SRAM, EPROM, NOR Flash EPROM, NAND Flash EPROM, burstable RAM, and other peripherals.

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

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

- Memory controller with eight memory banks (chip selects)
 - 32-bit address decoding with mask
 - Variable memory block sizes (32 Kbytes to 2 Gbytes in FCM mode, 32 Kbytes to 64 Mbytes in UPM mode, and 32 Kbytes to 64 Mbytes in GPCM mode)
 - Selection of control signal generation on a per-bank basis
 - Data buffer controls activated on a per-bank basis
 - Automatic segmentation of large transactions into memory accesses optimized for bus width and addressing capability
 - Write-protection capability
 - Atomic operation
- General-purpose chip-select machine (GPCM)
 - Compatible with SRAM, EPROM, NOR Flash EEPROM, FEPRM, and peripherals
 - Global (boot) chip-select available at system reset
 - Boot chip-select support for 8-, 16-bit devices
 - Minimum three-clock access to external devices
 - Two byte-write-enable signals ($\overline{\text{LWE}}[0:1]$)
 - Output enable signal ($\overline{\text{LOE}}$)
 - External access termination signal ($\overline{\text{LGTA}}$)
- NAND Flash control machine (FCM)
 - Compatible with small (512 + 16 bytes) and large (2048 + 64 bytes) page parallel NAND Flash EEPROM
 - Global (boot) chip-select available at system reset, with 4-Kbyte boot block buffer for execute-in-place boot loading
 - Boot chip-select support for 8-bit devices
 - Dual 2-Kbyte/eight 512-byte buffers allow simultaneous data transfer during Flash reads and programming
 - Interrupt-driven block transfer for reads and writes
 - Programmable command and data transfer sequences of up to eight steps supported
 - Generic command and address registers support proprietary Flash interfaces
 - Block write locking to ensure system security and integrity
- Three user-programmable machines (UPMs)
 - Programmable-array-based machine controls external signal timing with a granularity of up to one quarter of an external bus clock period
 - User-specified control-signal patterns run when an internal master requests a single-beat or burst read or write access
 - UPM refresh timer runs a user-specified control signal pattern to support refresh
 - User-specified control-signal patterns can be initiated by software

- Each UPM can be defined to support devices with depths of 64, 128, 256, and 512 Kbytes, and 1, 2, 4, 8, 16, 32, and 64 Mbytes
- Support for 8- and 16-bit devices
- Page mode support for successive transfers within a burst
- Optional monitoring of transfers between local bus internal masters and local bus slaves (local bus error reporting)

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

1.3.8 Enhanced Secure Digital Host Controller (eSDHC)

The enhanced secure digital host controller (eSDHC) provides an interface between the host system and these types of memory cards:

- MultiMediaCard (MMC)

MMC is a universal low-cost data storage and communication medium designed to cover a wide area of applications including mobile video and gaming, which are available from either pre-loaded MMC cards or downloadable from cellular phones, WLAN, or other wireless networks.
- Secure digital (SD) card

The secure digital (SD) card is an evolution of old MMC technology. It is specifically designed to meet the security, capacity, performance, and environment requirements inherent in the emerging

audio and video consumer electronic devices. The physical form factor, pin assignments, and data transfer protocol are backward-compatible with the old MMC.

- SDIO

Under the SD protocol, the SD cards can be categorized as a memory card, I/O card, or combo card. The memory card invokes a copyright protection mechanism that complies with the security of the SDMI standard. The I/O card provides high-speed data I/O with low power consumption for mobile electronic devices. The combo card has both memory and I/O functions.

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

The eSDHC can select the following modes for data transfer:

- SD 1-bit
- SD 4-bit
- MMC 1-bit
- MMC 4-bit
- Identification mode (up to 400 KHz)
- Full-speed mode (up to 25 MHz) or high-speed mode (up to 31.25 MHz)

The eSDHC includes the following features:

- Supports single- and multi-block read and write
- Supports write protection switch for write operations
- Supports synchronous and asynchronous abort
- Supports pause during the data transfer at a block gap
- Supports SDIO read wait and suspend/resume operations
- Supports Auto CMD12 for multi-block transfer
- Host can initiate non-data transfer commands while the data transfer is in progress
- Allows cards to interrupt the host in 1- and 4-bit SDIO modes
- Embodies a fully configurable 128 × 32-bit FIFO for read/write data
- Built-in DMA capabilities

1.3.9 Universal Serial Bus (USB) 2.0

The USB 2.0 controller offers operation as a host or device. The USB controller provides point-to-point connectivity, which complies with the *Universal Serial Bus Revision 2.0 Specification*. The USB controllers can be configured to operate as a stand-alone host or stand-alone device. See [Figure 1-4](#) for more information.

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

- Bulk
- Control

- Interrupt
- Isochronous

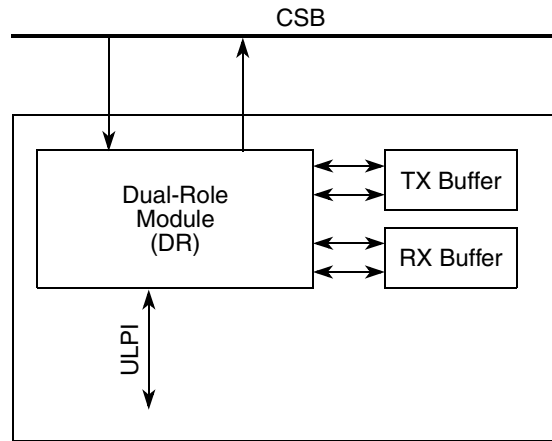


Figure 1-4. USB Controllers Port Configuration

1.3.9.1 USB Dual-Role Controller

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

1.3.10 FlexCAN Module

- Full implementation of the CAN protocol specification version 2.0B
 - Standard data and remote frames (up to 109 bits long)
 - Extended data and remote frames (up to 127 bits long)
 - 0–8 bytes data length
 - Programmable bit rate up to 1 Mbps
 - Content-related addressing
- Up to 64 flexible message buffers of zero to eight bytes data length, each configurable as Rx or Tx, all supporting standard and extended messages
- Individual mask registers for each message buffer

- Includes 1056 bytes of RAM (254×4) for 64 message buffers storage
- Includes 256 bytes of RAM (64×4) for 64 individual Rx mask registers
- Full featured Rx FIFO with storage capacity for six frames and internal pointer handling
- Unused message buffer and Rx mask register space can be used as general purpose RAM space
- Listen only mode capability
- Programmable transmission priority scheme: lowest ID or lowest buffer number
- Time stamp based on 16-bit, free-running timer
- Global network time, synchronized by a specific message
- Maskable interrupts
- Independent of the transmission medium (an external transceiver is required)
- Short latency time due to an arbitration scheme for high-priority messages

1.3.11 Dual I²C Interfaces

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 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 I²C allows the connection of additional devices to the bus for expansion and system development.

The I²C 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 I²C controller consists of a transmitter/receiver unit, clocking unit, and control unit. The I²C unit supports general broadcast mode and on-chip filtering rejects spikes on the bus.

The I²C 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
- I²C 1 can be used as the boot sequencer

1.3.12 DMA Engine 1

The DMA (direct memory access) is capable of performing complex data transfers with minimal intervention from a host processor via two programmable channels. The hardware architecture includes a

DMA engine which performs source and destination address calculations, and the actual data movement operations, along with a local memory containing the transfer control descriptors (TCD) for the channels. This SRAM-based implementation is utilized to minimize the overall module size.

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

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

1.3.13 DMA Engine 2

The DMA Engine 2 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. 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. The DMA Engine 2 has 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

1.3.14 Dual Universal Asynchronous Receiver/Transmitter (DUART)

MPC8309 includes two DUARTs intended for use in maintenance, bring up, and debug systems. The device provides a standard two-wire data (TXD, RXD) for each port (can also be configured as one 4-wire interface (RxD, TxD, RTS, CTS)). The DUART is a slave interface. An interrupt is provided to the interrupt controller. Interrupts are generated for transmit, receive, and line status.

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

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

The DUART includes the following features:

- Full-duplex operation

- Programming model compatible with the original PC16450 UART and the PC16550D (an improved version of the PC16450 that also operates in FIFO mode)
- PC16450 register reset values
- FIFO mode for both transmitter and receiver, providing 16-byte FIFOs
- Serial data encapsulation and decapsulation with standard asynchronous communication bits (START, STOP, and parity)
- Maskable transmit, receive, and line status interrupts
- Software-programmable baud rate generators that divide the system clock by 1 to $(2^{16} - 1)$ and generate a 16x clock for the transmitter and receiver engines
- Software-selectable serial-interface data format (data length, parity, 1/1.5/2 STOP bit, baud rate)
- Line status registers
- Line-break detection and generation
- Internal diagnostic support, local loopback, and break functions
- Prioritized interrupt reporting
- Overrun, parity, and framing error detection

1.3.15 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.

1.3.16 System Timers

The system includes the following timers:

- Periodic interrupt timer
- 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

1.3.17 Real Time Clock

RTC has the following features:

- Maintains a one-second count, unique over a period of thousand of years
- A 32-bit counter, which:
 - Increments for every one second
 - Can be initialized by software to a specific initial count value
- An alarm function with programmable and maskable alarm interrupt

- Programmable and maskable every second interrupt
- Two possible clock sources:
 - External RTC clock (RTC_PIT_CLK)
 - CSB bus clock
- RTC function can be disabled, if required

Chapter 2 Memory Map

This chapter describes the MPC8309 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 2-Mbyte address region. To allow for flexibility, the base address of the memory mapped registers is re-locatable in the local address space. The local address map location of this register block is controlled by the internal memory mapped registers base address register (IMMRBAR). See [Section 6.2.4.1, “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 IMMR Address Map

[Table 2-1](#) lists the location of the functional block base addresses for the entire IMMRBAR space. Unless stated otherwise in a particular block, all accesses to and from the memory mapped registers must be made with 32-bit accesses. There is no support for accesses of sizes other than 32 bits.

Reading from address locations 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 are read as zero unless the reset value of those bits is different due to internal logic considerations.

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

Memory Map

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

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

Table 2-1. IMMR Memory Map

Block Base Address	Block	Actual Size	Window	Section/ Page
0x0_0000–0x0_01FF	System configuration	512 bytes	512 bytes	6.3.1/6-16
0x0_0200–0x0_02FF	Watchdog timer	16 bytes	256 bytes	6.4.4/6-41
0x0_0300–0x0_03FF	Real time clock	32 bytes	256 bytes	6.5.5/6-48
0x0_0400–0x0_04FF	Periodic interval timer	32 bytes	256 bytes	6.6.5/6-55
0x0_0500–0x0_05FF	Global timers module 1	64 bytes	256 bytes	6.7.5/6-64
0x0_0600–0x0_06FF	Reserved	—	256 bytes	—
0x0_0700–0x0_07FF	Integrated programmable interrupt controller (IPIC)	128 bytes	256 bytes	9.5/9-5
0x0_0800–0x0_08FF	System arbiter	30 bytes	256 bytes	7.2/7-2
0x0_0900–0x0_09FF	Reset module	44 bytes	256 bytes	4.5.1/4-27
0x0_0A00–0x0_0AFF	Clock module	44 bytes	256 bytes	4.5.2/4-31
0x0_0B00–0x0_0BFF	Power management control module	20 bytes	256 bytes	6.8.2/6-77
0x0_0C00–0x0_0CFF	GPIO 1	24 bytes	256 bytes	21.3/21-2
0x0_0D00–0x0_0DFF	GPIO 2	24 bytes	256 bytes	21.3/21-2
0x0_0E00–0x0_12FF	Reserved	—	1.25 Kbytes	—
0x0_1300–0x0_13FF	QUICC Engine port interrupts	20 bytes	256 bytes	9.5.24/9-32
0x0_1400–0x0_17FF	Reserved	—	1 Kbyte	—
0x0_1800–0x0_1FFF	Reserved	—	2 Kbytes	—
0x0_2000–0x0_2FFF	DDR memory controller	3.8 Kbytes	4 Kbytes	10.4/10-9
0x0_3000–0x0_30FF	I ² C controller 1	24 bytes	256 bytes	17.3/17-4
0x0_3100–0x0_31FF	I ² C controller 2	24 bytes	256 bytes	17.3/17-4
0x0_3200–0x0_44FF	Reserved	—	4.75 Kbytes	—
0x0_4500–0x0_46FF	DUART1 (UART1 and UART2)	18 bytes x 2	512 bytes	18.3/18-4
0x0_4700–0x0_48FF	Reserved	—	512 bytes	—
0x0_4900–0x0_4AFF	DUART2 (UART3 and UART4)	18 bytes x 2	512 bytes	18.3/18-4

Table 2-1. IMMR Memory Map (continued)

Block Base Address	Block	Actual Size	Window	Section/ Page
0x0_4B00–0x0_4FFF	Reserved	—	1.25 Kbytes	—
0x0_5000–0x0_5FFF	eLBC	224 bytes	4 Kbytes	11.3/11-8
0x0_6000–0x0_6FFF	Reserved	—	4 Kbytes	—
0x0_7000–0x0_70FF	SPI	24 bytes	256 bytes	19.4/19-8
0x0_7100–0x0_7FFF	Reserved	—	3.75 Kbytes	—
0x0_8000–0x0_82FF	DMA Engine 2	680 bytes	768 bytes	14.2/14-2
0x0_8300–0x0_83FF	Reserved	—	256 bytes	—
0x0_8400–0x0_84FF	I/O sequencer	256 bytes	256 bytes	22.3/22-2
0x0_8500–0x1_BFFF	Reserved	—	78.75 Kbytes	—
0x1_C000–0x1_CFFF	FlexCAN 1	1 Kbyte	4 Kbytes	15.3/15-5
0x1_D000–0x1_DFFF	FlexCAN 2	1 Kbyte	4 Kbytes	15.3/15-5
0x1_E000–0x2_2FFF	Reserved	—	20 Kbytes	—
0x2_3000–0x2_3FFF	USB DR (Device/ OTG)	1280 bytes	4 Kbytes	16.3/16-4
0x2_4000–0x2_8FFF	Reserved	—	20 Kbytes	—
0x2_9000–0x2_9FFF	FlexCAN 3	1 Kbyte	4 Kbytes	15.3/15-5
0x2_A000–0x2_AFFF	FlexCAN 4	1 Kbyte	4 Kbytes	15.3/15-5
0x2_B000–0x2_BFFF	Reserved	—	4 Kbytes	—
0x2_C000–0x2_DFFF	DMA Engine 1	8 Kbytes	8 kbytes	13.2/13-2
0x2_E000–0x2_EFFF	eSDHC	4 Kbytes	4 Kbytes	12.4/12-5
0x2_F000–0xF_FFFF	Reserved	—	836 Kbytes	—
0x10_0000–0x1F_FFFF	QUICC Engine	1 Mbytes	1 Mbytes	9.5.24/9-32
0x20_0000–0xFF_FFFF	Reserved	—	14 Mbytes	—



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 a list ordered by functional block and a list by alphabetical order.
- List of output signal states at reset.

NOTE

A bar over a signal name indicates that the signal is active low, such as $\overline{\text{MWE}}$. Active-low signals are referred to as asserted (active) when they are low and negated when they are high. Signals that are active high, such as FEC3_RX_DV , are referred to as asserted when they are high and negated when they are low.

3.1 Signals Overview

The signals are grouped as follows:

- DDR2 memory interface signals
- DUART interface signals
- I²C interface signals
- QE Ethernet management signals
- QE FEC signals
- QE TDM/HDLC signals
- eLBC signals
- GPIO interface signals
- Global timers
- USB signals
- PCI signals
- IEEE 1588 signals
- IPIC interface signals
- SPI interface signals
- JTAG interface signals
- FlexCAN interface signals
- System control signals
- Test interface signals
- Clock interface signals
- eSDHC interface signals
- Miscellaneous signals

Figure 3-1, Figure 3-2, and Figure 3-3 show the external signals of the device and how the signals are grouped for the MPC8309. Refer to the *MPC8309 PowerQUICC II Pro Integrated Communications Processor Hardware Specification* for a pinout diagram showing pin numbers and a listing of all the electrical and mechanical specifications.

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

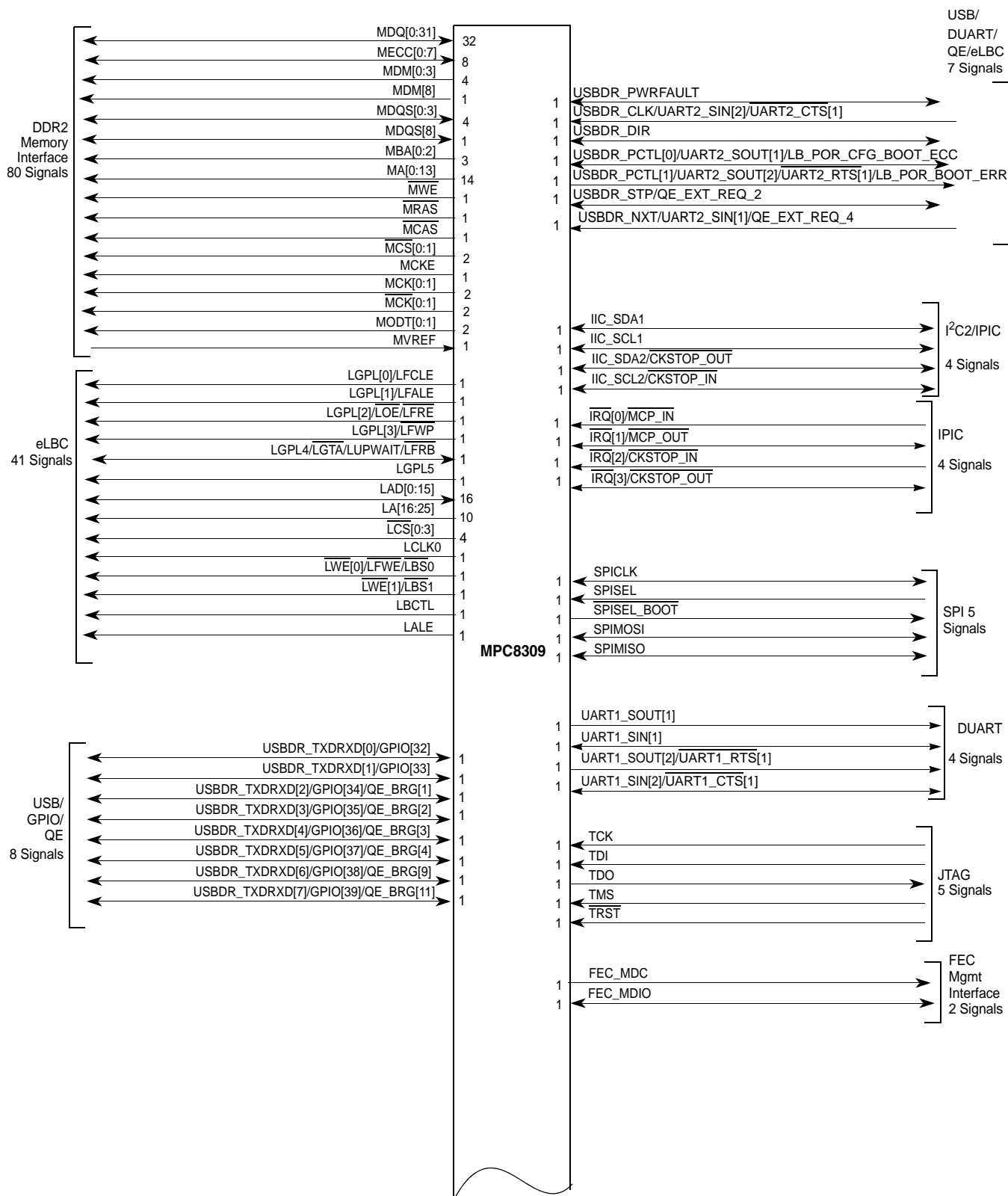


Figure 3-1. MPC8309 Signal Groupings (1 of 3)

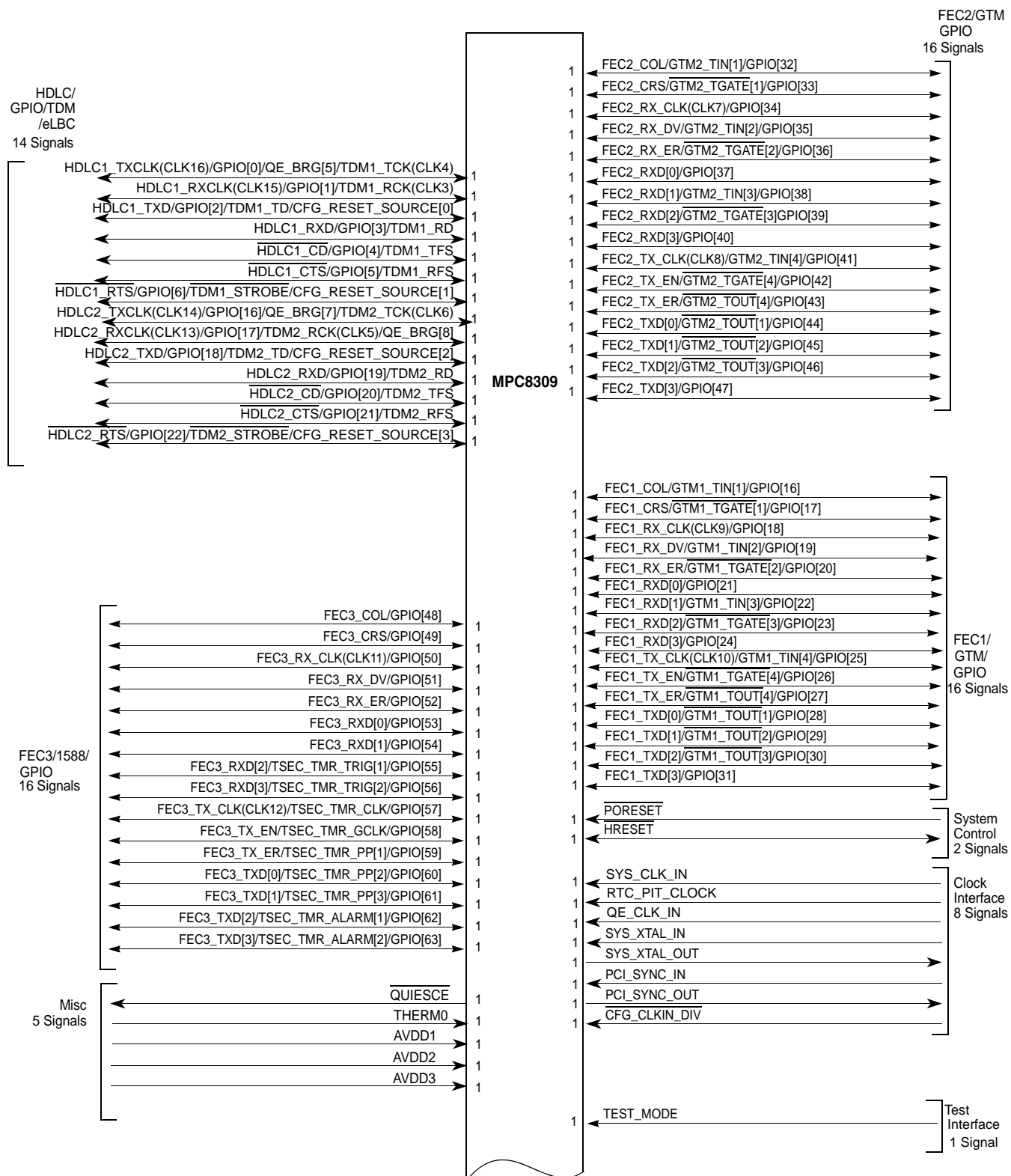


Figure 3-2. MPC8309 Signal Groupings (2 of 3)

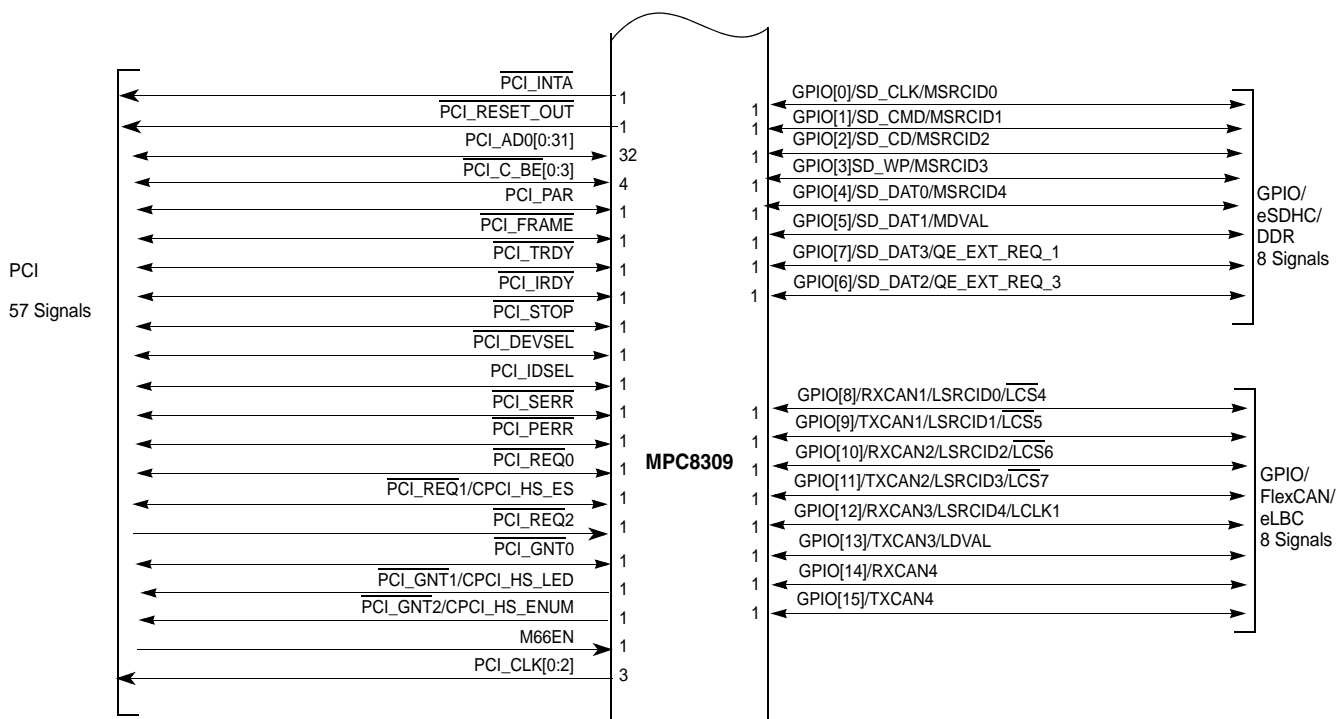


Figure 3-3. MPC8309 Signal Groupings (3 of 3)

The following tables provide summaries of MPC8309 signal functions. [Table 3-1](#) provides a summary of 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. MPC8309 Signal Reference by Functional Block

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/Page
MA[0:13]	DDR address	DDR2	—	14	O	10.3/10-3
MECC[0:7]	DDR ECC bits	DDR2	—	8	I/O	10.3/10-3
MBA[0:2]	DDR bank select	DDR2	—	3	O	10.3/10-3
MCAS	DDR column address strobe	DDR2	—	1	O	10.3/10-3
MCK[0:1]	DDR differential clocks	DDR2	—	2	O	10.3/10-3
MCK[0:1]	DDR differential clocks	DDR2	—	2	O	10.3/10-3
MCKE	DDR clock enable	DDR2	—	1	O	10.3/10-3
MCS[0:1]	DDR chip select	DDR2	—	2	O	10.3/10-3

Table 3-1. MPC8309 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/ Page
MDM[0:3]	DDR data mask	DDR2	—	4	O	10.3/10-3
MDM[8]	DDR data mask	DDR2	—	1	O	10.3/10-3
MDQ[0:31]	DDR data	DDR2	—	32	I/O	10.3/10-3
MDQS[0:3]	DDR data strobe	DDR2	—	4	I/O	10.3/10-3
MDQS[8]	DDR data strobe	DDR2	—	1	I/O	10.3/10-3
MODT[0:1]	DDR on-die termination	DDR2	—	2	O	10.3/10-3
$\overline{\text{MRAS}}$	DDR row address strobe	DDR2	—	1	O	10.3/10-3
MVREF	DDR DRAM reference	DDR2	—	1	PWR	10.3/10-3
$\overline{\text{MWE}}$	DDR write enable	DDR2	—	1	O	10.3/10-3
LA[16:25]	LBC port address	eLBC	—	10	O	11.2/11-4
LAD[0:15]	LBC address data	eLBC	—	16	I/O	11.2/11-4
LALE	LBC address latch enable	eLBC	—	1	O	11.2/11-4
LB_POR_CFG_BOOT_ECC	Enable/disable ECC for flash during RCW load	eLBC	USBDP_PCTL[0]/ UART2_SOUT[1]	1	I/O	11.2/11-4
LB_POR_BOOT_ERR	Status of ECC error during boot loading from flash	eLBC	USBDP_PCTL[1]/ UART2_SOUT[2]/ UART2_RTS[1]	1	O	11.2/11-4
LBCTL	LBC data buffer control	eLBC	—	1	O	11.2/11-4
LCLK0	LBC clock 0	eLBC	—	1	I/O	11.2/11-4
LCLK1	LBC clock 1	eLBC	GPIO[12]/RXCAN3/ LSRCID4	1	I/O	11.2/11-4
$\overline{\text{LCS}}[0:3]$	LBC chip select 0–3	eLBC	—	4	O	11.2/11-4
$\overline{\text{LCS}}[4]$	LBC chip select 4	eLBC	GPIO[8]/RXCAN1/ LSRCID[0]	1	O	11.2/11-4
$\overline{\text{LCS}}[5]$	LBC chip select 5	eLBC	GPIO[9]/TXCAN1/ LSRCID[1]	1	I/O	11.2/11-4
$\overline{\text{LCS}}[6]$	LBC chip select 6	eLBC	GPIO[10]/RXCAN2/ LSRCID[2]	1	O	11.2/11-4
$\overline{\text{LCS}}[7]$	LBC chip select 7	eLBC	GPIO[11]/TXCAN2/ LSRCID[3]	1	I/O	11.2/11-4
LGPL[0]/LFCLE	LBC UPM general purpose line 0	eLBC	—	1	O	11.2/11-4
LGPL[1]/LFALE	LBC UPM general purpose line 1	eLBC	—	1	O	11.2/11-4
LGPL[2]/ $\overline{\text{LOE}}$ /LFRE	LBC UPM general purpose line 2	eLBC	—	1	O	11.2/11-4

Table 3-1. MPC8309 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/ Page
LGPL[3]/LFWP	LBC UPM general purpose line 3	eLBC	—	1	O	11.2/11-4
LGPL4/LGTA/LUPWAIT/LFRB	LBC UPM general purpose line 4	eLBC	—	1	I/O	11.2/11-4
LGPL5	LBC UPM general purpose line 5	eLBC	—	1	O	11.2/11-4
LWE[0]/LFWELBS0	LBC write enable 0	eLBC	—	1	O	11.2/11-4
LWE[1]/LBS1	LBC write enable 1	eLBC	—	1	O	11.2/11-4
LSRCID[0]	Local bus debug source ID 0	LBC	GPIO[8]/RXCAN1/LCS4	1	I/O	11.2/11-4
LSRCID[1]	Local bus debug source ID 1	LBC	GPIO[9]/TXCAN1/LCS5	1	I/O	11.2/11-4
LSRCID[2]	Local bus debug source ID 2	LBC	GPIO[10]/RXCAN2/LCS6	1	I/O	11.2/11-4
LSRCID[3]	Local bus debug source ID 3	LBC	GPIO[11]/TXCAN2/LCS7	1	I/O	11.2/11-4
LSRCID[4]	Local bus debug source ID 4	LBC	GPIO[12]/RXCAN3/LCLK1	1	O	11.2/11-4
LDVAL	Local bus debug data valid	LBC	GPIO[13]/TXCAN3	1	I/O	11.2/11-4
UART1_CTS[1]	DUART clear to send 1	DUART	UART1_SIN[2]	1	I/O	18.2/18-3
UART1_RTS[1]	DUART ready to send 1	DUART	UART1_SOUT[2]	1	O	18.2/18-3
UART1_SIN[1]	DUART serial data in 1	DUART	—	1	I/O	18.2/18-3
UART1_SIN[2]	DUART serial data in 2	DUART	UART1_CTS[1]	1	I/O	18.2/18-3
UART1_SOUT[1]	DUART serial data out 1	DUART	—	1	O	18.2/18-3
UART1_SOUT[2]	DUART serial data out 2	DUART	UART1_RTS[1]	1	O	18.2/18-3
UART2_CTS[1]	DUART clear to send 1	DUART	USBDR_CLK/ UART2_SIN[2]	1	I	18.2/18-3
UART2_RTS[1]	DUART ready to send 1	DUART	USBDR_PCTL[1]/ UART2_SOUT[2]/ LB_POR_BOOT_ERR	1	O	18.2/18-3
UART2_SIN[1]	DUART serial data in 1	DUART	USBDR_NXT/QE_EXT_REQ_4	1	I/O	18.2/18-3
UART2_SIN[2]	DUART serial data in 2	DUART	USBDR_CLK/ UART2_CTS[1]	1	I	18.2/18-3
UART2_SOUT[1]	DUART serial data out 1	DUART	USBDR_PCTL[0]/ LB_POR_CFG_BOOT_ECC	1	I/O	18.2/18-3

Table 3-1. MPC8309 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/ Page
UART2_SOUT[2]	DUART serial data out 2	DUART	USBDP_PCTL[1]/ UART2_RTS[1]/ LB_POR_BOOT_ ERR	1	O	18.2/18-3
QE_CLK_IN	Reference clock input for QE PLL	Clocks	—	1	I	4.1.2/4-2
RTC_PIT_CLOCK	Clock input for RTC and PIT blocks (32.768 KHz clock is commonly used)	Clocks	—	1	I	4.1.2/4-2
SYS_CLK_IN	Reference input clock for system PLL	Clocks	—	1	I	4.1.2/4-2
SYS_XTAL_IN	Crystal clock input	Clocks	—	1	I	4.1.2/4-2
SYS_XTAL_OUT	Crystal clock output	Clocks	—	1	O	4.1.2/4-2
PCI_SYNC_IN	PCI clock sync input	Clocks	—	1	I	4.1.2/4-2
PCI_SYNC_OUT	PCI clock sync output	Clocks	—	1	O	4.1.2/4-2
CFG_CLKIN_DIV	Configuration clock in division selection	Reset and clock	—	1	I	4.1.2/4-2
SD_CD	Card detection signal	eSDHC	GPIO[2]/MSRCID[2]	1	I/O	12.3/12-4
SD_CLK	eSDHC clock out	eSDHC	GPIO[0]/MSRCID[0]	1	I/O	12.3/12-4
SD_CMD	eSDHC command/response signal	eSDHC	GPIO[1]/MSRCID[1]	1	I/O	12.3/12-4
SD_DAT[0]	Data signal 0	eSDHC	GPIO[4]/MSRCID[4]	1	I/O	12.3/12-4
SD_DAT[1]	Data signal1	eSDHC	GPIO[5]/MDVAL	1	I/O	12.3/12-4
SD_DAT[2]	Data signal2	eSDHC	GPIO[6]/ QE_EXT_REQ_3	1	I/O	12.3/12-4
SD_DAT[3]	Data signal3	eSDHC	GPIO[7]/ QE_EXT_REQ_1	1	I/O	12.3/12-4
SD_WP	Write protection signal	eSDHC	GPIO[3]/ MSRCID[3]	1	I/O	12.3/12-4
PCI_INTA	PCI interrupt output	PCI	—	1	O	23.2/23-4
PCI_RESET_OUT	PCI reset	PCI	—	1	O	23.2/23-4
PCI_AD[0:31]	PCI address/data 0	PCI	—	32	I/O	23.2/23-4
PCI_C_BE[0:3]	PCI command/byte enable 0:4	PCI	—	4	I/O	23.2/23-4
PCI_PAR	PCI parity	PCI	—	1	I/O	23.2/23-4
PCI_FRAME	PCI frame	PCI	—	1	I/O	23.2/23-4
PCI_TRDY	PCI target ready	PCI	—	1	I/O	23.2/23-4

Table 3-1. MPC8309 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/ Page
$\overline{\text{PCI_IRDY}}$		PCI	—	1	I/O	23.2/23-4
$\overline{\text{PCI_STOP}}$	PCI stop	PCI	—	1	I/O	23.2/23-4
$\overline{\text{PCI_DEVSEL}}$	PCI device select	PCI	—	1	I/O	23.2/23-4
$\overline{\text{PCI_IDSEL}}$	PCI initial device select	PCI	—	1	I/O	23.2/23-4
$\overline{\text{PCI_SERR}}$	PCI system error	PCI	—	1	I/O	23.2/23-4
$\overline{\text{PCI_PERR}}$	PCI parity error	PCI	—	1	I/O	23.2/23-4
$\overline{\text{PCI_REQ0}}$	PCI request 0	PCI	—	1	I/O	23.2/23-4
$\overline{\text{PCI_REQ1}}$	PCI request 1	PCI	CPCI_HS_ES	1	I/O	23.2/23-4
$\overline{\text{PCI_REQ2}}$	PCI request 2	PCI	—	1	I	23.2/23-4
$\overline{\text{PCI_GNT0}}$	PCI grant 0	PCI	—	1	I/O	23.2/23-4
$\overline{\text{PCI_GNT1}}$	PCI grant 1	PCI	CPCI_HS_LED	1	O	23.2/23-4
$\overline{\text{PCI_GNT2}}$	PCI grant 2	PCI	CPCI_HS_ENUM	1	O	23.2/23-4
M66EN	66-MHz system configuration	PCI	—	1	I	23.2/23-4
$\text{PCI_CLK}[0:2]$	PCI clock 0:2	PCI	—	3	O	23.2/23-4
CPCI_HS_ES	Compact PCI hot-swap ejector switch	PCI	$\overline{\text{PCI_REQ1}}$	1	I/O	23.2/23-4
CPCI_HS_LED	Compact PCI hot-swap LED	PCI	$\overline{\text{PCI_GNT1}}$	1	O	23.2/23-4
CPCI_HS_ENUM	Compact PCI hot-swap enumerator	PCI	$\overline{\text{PCI_GNT2}}$	1	O	23.2/23-4
SPICLK	SPI clock	SPI	—	1	I/O	19.3/19-6
SPIMISO	SPI master-in slave-out	SPI	—	1	I/O	19.3/19-6
SPIMOSI	SPI master-out slave-in	SPI	—	1	I/O	19.3/19-6
SPISEL	SPI slave select	SPI	—	1	I/O	19.3/19-6
$\overline{\text{SPISEL_BOOT}}$	SPI slave select boot	SPI	—	1	O	19.3/19-6
$\overline{\text{IRQ}}[0]$	External interrupt 0/ Machine check interrupt input	IPIC	$\overline{\text{MCP_IN}}$	1	I	9.4/9-5
$\overline{\text{IRQ}}[1]$	External interrupt 1	IPIC	$\overline{\text{MCP_OUT}}$	1	I/O	9.4/9-5
$\overline{\text{IRQ}}[2]$	External interrupt 2	IPIC	$\overline{\text{CKSTOP_IN}}$	1	I	9.4/9-5
$\overline{\text{IRQ}}[3]$	External interrupt 3	IPIC	$\overline{\text{CKSTOP_OUT}}$	1	I/O	9.4/9-5
$\overline{\text{MCP_OUT}}$	Machine check interrupt output	IPIC	$\overline{\text{IRQ}}[1]$	1	I/O	9.4/9-5
$\overline{\text{MCP_IN}}$	Machine check interrupt input	IPIC	$\overline{\text{IRQ}}[0]$	1	I/O	9.4/9-5

Table 3-1. MPC8309 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/ Page
TCK	Test clock	JTAG	—	1	I	20.2/20-1
TDI	Test data in	JTAG	—	1	I	20.2/20-1
TDO	Test data out	JTAG	—	1	O	20.2/20-1
TMS	Test mode select	JTAG	—	1	I	20.2/20-1
$\overline{\text{TRST}}$	Test reset	JTAG	—	1	I	20.2/20-1
FEC_MDC	FEC management data clock	FEC	—	1	O	6.3.2/6-17
FEC_MDIO	FEC management data in/out	FEC	—	1	I/O	6.3.2/6-17
FEC1_COL	FEC1 collision detect	FEC1	GTM1_TIN[1]/GPIO[16]	1	I/O	6.3.2/6-17
FEC1_CRIS	FEC1 carrier case	FEC1	$\overline{\text{GTM1_TGATE}}[1]/$ GPIO[17]	1	I/O	6.3.2/6-17
FEC1_RX_CLK(CLK9)	FEC1 receive clock	FEC1	GPIO[18]	1	I/O	6.3.2/6-17
FEC1_RX_DV	FEC1 receive data valid	FEC1	GTM1_TIN[2]/GPIO[19]	1	I/O	6.3.2/6-17
FEC1_RX_ER	FEC1 receive error	FEC1	$\overline{\text{GTM1_TGATE}}[2]/$ GPIO[20]	1	I/O	6.3.2/6-17
FEC1_RXD[0]	FEC1 receive data 0	FEC1	GPIO[21]	1	I/O	6.3.2/6-17
FEC1_RXD[1]	FEC1 receive data 1	FEC1	GTM1_TIN[3]/ GPIO[22]	1	I/O	6.3.2/6-17
FEC1_RXD[2]	FEC1 receive data 2	FEC1	$\overline{\text{GTM1_TGATE}}[3]/$ GPIO[23]	1	I/O	6.3.2/6-17
FEC1_RXD[3]	FEC1 receive data 3	FEC1	GPIO[24]	1	I/O	6.3.2/6-17
FEC1_TX_CLK (CLK10)	FEC1 transmit clock in	FEC1	GTM1_TIN[4]/ GPIO[25]	1	I/O	6.3.2/6-17
FEC1_TX_EN	FEC1 transmit enable	FEC1	$\overline{\text{GTM1_TGATE}}[4]/$ GPIO[26]	1	I/O	6.3.2/6-17
FEC1_TX_ER	FEC1 transmit error	FEC1	$\overline{\text{GTM1_TOUT}}[4]/$ GPIO[27]	1	I/O	6.3.2/6-17
FEC1_TXD[0]	FEC1 transmit data 0	FEC1	$\overline{\text{GTM1_TOUT}}[1]/$ GPIO[28]	1	I/O	6.3.2/6-17
FEC1_TXD[1]	FEC1 transmit data 1	FEC1	$\overline{\text{GTM1_TOUT}}[2]/$ GPIO[29]	1	I/O	6.3.2/6-17
FEC1_TXD[2]	FEC1 transmit data 2	FEC1	$\overline{\text{GTM1_TOUT}}[3]/$ GPIO[30]	1	I/O	6.3.2/6-17
FEC1_TXD[3]	FEC1 transmit data 3	FEC1	GPIO[31]	1	I/O	6.3.2/6-17
FEC2_COL	FEC2 collision detect	FEC2	GTM2_TIN[1]/GPIO[32]	1	I/O	6.3.2/6-17

Table 3-1. MPC8309 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/ Page
FEC2_CRS	FEC2 carrier case	FEC2	$\overline{\text{GTM2_TGATE}}[1]/$ GPIO[33]	1	I/O	6.3.2/6-17
FEC2_RX_CLK (CLK7)	FEC2 receive clock	FEC2	GPIO[34]	1	I/O	6.3.2/6-17
FEC2_RX_DV	FEC2 receive data valid	FEC2	GTM2_TIN[2]/ GPIO[35]	1	I/O	6.3.2/6-17
FEC2_RX_ER	FEC2 receive error	FEC2	$\overline{\text{GTM2_TGATE}}[2]/$ GPIO[36]	1	I/O	6.3.2/6-17
FEC2_RXD[0]	FEC2 receive data 0	FEC2	GPIO[37]	1	I/O	6.3.2/6-17
FEC2_RXD[1]	FEC2 receive data 1	FEC2	GTM2_TIN[3]/ GPIO[38]	1	I/O	6.3.2/6-17
FEC2_RXD[2]	FEC2 receive data 2	FEC2	$\overline{\text{GTM2_TGATE}}[3]/$ GPIO[39]	1	I/O	6.3.2/6-17
FEC2_RXD[3]	FEC2 receive data 3	FEC2	GPIO[40]	1	I/O	6.3.2/6-17
FEC2_TX_CLK (CLK8)	FEC2 transmit clock in	FEC2	GTM2_TIN[4]/ GPIO[41]	1	I/O	6.3.2/6-17
FEC2_TX_EN	FEC2 transmit enable	FEC2	$\overline{\text{GTM2_TGATE}}[4]/$ GPIO[42]	1	I/O	6.3.2/6-17
FEC2_TX_ER	FEC2 transmit error	FEC2	GTM2_TOUT[4]/ GPIO[43]	1	I/O	6.3.2/6-17
FEC2_TXD[0]	FEC2 transmit data 0	FEC2	GTM2_TOUT[1]/ GPIO[44]	1	I/O	6.3.2/6-17
FEC2_TXD[1]	FEC2 transmit data 1	FEC2	GTM2_TOUT[2]/ GPIO[45]	1	I/O	6.3.2/6-17
FEC2_TXD[2]	FEC2 transmit data 2	FEC2	GTM2_TOUT[3]/ GPIO[46]	1	I/O	6.3.2/6-17
FEC2_TXD[3]	FEC2 transmit data 3	FEC2	GPIO[47]	1	I/O	6.3.2/6-17
FEC3_COL	FEC3 collision detect	FEC3	GPIO[48]	1	I/O	6.3.2/6-17
FEC3_CRS	FEC3 carrier case	FEC3	GPIO[49]	1	I/O	6.3.2/6-17
FEC3_RX_CLK (CLK11)	FEC3 receive clock	FEC3	GPIO[50]	1	I/O	6.3.2/6-17
FEC3_RX_DV	FEC3 receive data valid	FEC3	GPIO[51]	1	I/O	6.3.2/6-17
FEC3_RX_ER	FEC3 receive error	FEC3	GPIO[52]	1	I/O	6.3.2/6-17
FEC3_RXD[0]	FEC3 receive data 0	FEC3	GPIO[53]	1	I/O	6.3.2/6-17
FEC3_RXD[1]	FEC3 receive data 1	FEC3	GPIO[54]	1	I/O	6.3.2/6-17
FEC3_RXD[2]	FEC3 receive data 2	FEC3	TSEC_TMR_TRIG[1]/ GPIO[55]	1	I/O	6.3.2/6-17

Table 3-1. MPC8309 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/ Page
FEC3_RXD[3]	FEC3 receive data 3	FEC3	TSEC_TMR_TRIG[2]/ GPIO[56]	1	I/O	6.3.2/6-17
FEC3_TX_CLK (CLK12)	FEC3 transmit clock in	FEC3	TSEC_TMR_CLK/ GPIO[57]	1	I/O	6.3.2/6-17
FEC3_TX_EN	FEC3 transmit enable	FEC3	TSEC_TMR_GCLK/ GPIO[58]	1	I/O	6.3.2/6-17
FEC3_TX_ER	FEC3 transmit error	FEC3	TSEC_TMR_PP[1]/ GPIO[59]	1	I/O	6.3.2/6-17
FEC3_TXD[0]	FEC3 transmit data 0	FEC3	TSEC_TMR_PP[2]/ GPIO[60]	1	I/O	6.3.2/6-17
FEC3_TXD[1]	FEC3 transmit data 1	FEC3	TSEC_TMR_PP[3]/ GPIO[61]	1	I/O	6.3.2/6-17
FEC3_TXD[2]	FEC3 transmit data 2	FEC3	TSEC_TMR_ALARM[1]/ GPIO[62]	1	I/O	6.3.2/6-17
FEC3_TXD[3]	FEC3 transmit data 3	FEC3	TSEC_TMR_ALARM[2]/ GPIO[63]	1	I/O	6.3.2/6-17
RXCAN1	FlexCAN 1 receive data	FlexCAN	GPIO[8]/LSRCID0/ $\overline{\text{LCS}}_4$	1	I/O	15.2/15-5
TXCAN1	FlexCAN 1 transmit data	FlexCAN	GPIO[9]/LSRCID1/ $\overline{\text{LCS}}_5$	1	I/O	15.2/15-5
RXCAN2	FlexCAN 2 receive data	FlexCAN	GPIO[10]/LSRCID2/ $\overline{\text{LCS}}_6$	1	I/O	15.2/15-5
TXCAN2	FlexCAN 2 transmit data	FlexCAN	GPIO[11]/LSRCID3/ $\overline{\text{LCS}}_7$	1	I/O	15.2/15-5
RXCAN3	FlexCAN 3 receive data	FlexCAN	GPIO[12]/LSRCID4/ LCLK1	1	I/O	15.2/15-5
TXCAN3	FlexCAN 3 transmit data	FlexCAN	GPIO[13]/TXCAN3/ LDVAL	1	I/O	15.2/15-5
RXCAN4	FlexCAN 4 receive data	FlexCAN	GPIO[14]	1	I/O	15.2/15-5
TXCAN4	FlexCAN 4 transmit data	FlexCAN	GPIO[15]	1	I/O	15.2/15-5
$\overline{\text{GTM1_TGATE}}[1]$	Timer gate 1	Global timers	FEC1_CRG/GPIO[17]	1	I/O	6.7.4/6-62
$\overline{\text{GTM1_TGATE}}[2]$	Timer gate 2	Global timers	FEC1_RX_ER/GPIO[20]	1	I/O	6.7.4/6-62
$\overline{\text{GTM1_TGATE}}[3]$	Timer gate 3	Global timers	FEC1_RXD[2]/GPIO[23]	1	I/O	6.7.4/6-62
$\overline{\text{GTM1_TGATE}}[4]$	Timer gate 4	Global timers	FEC_TX_EN/GPIO[26]	1	I/O	6.7.4/6-62
GTM1_TIN[1]	Timer in 1	Global timers	FEC1_COL/GPIO[16]	1	I/O	6.7.4/6-62
GTM1_TIN[2]	Timer in 2	Global timers	FEC1_RX_DV/GPIO[19]	1	I/O	6.7.4/6-62
GTM1_TIN[3]	Timer in 3	Global timers	FEC1_RXD[1]/GPIO[22]	1	I/O	6.7.4/6-62

Table 3-1. MPC8309 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/ Page
GTM1_TIN[4]	Timer in 4	Global timers	FEC1_TX_CLK(CLK10)/GPIO[25]	1	I/O	6.7.4/6-62
GTM1_TOUT[1]	Timer out 1	Global timers	FEC1_TXD[0]/GPIO[28]	1	I/O	6.7.4/6-62
GTM1_TOUT[2]	Timer out 2	Global timers	FEC1_TXD[1]/GPIO[29]	1	I/O	6.7.4/6-62
GTM1_TOUT[3]	Timer out 3	Global timers	FEC1_TXD[2]/GPIO[30]	1	I/O	6.7.4/6-62
GTM1_TOUT[4]	Timer out 4	Global timers	FEC1_TX_ER/GPIO[27]	1	I/O	6.7.4/6-62
GTM2_TGATE[1]	Timer gate 1	Global timers	FEC2_CRG/GPIO[33]	1	I/O	6.7.4/6-62
GTM2_TGATE[2]	Timer gate 2	Global timers	FEC2_RX_ER/GPIO[36]	1	I/O	6.7.4/6-62
GTM2_TGATE[3]	Timer gate 3	Global timers	FEC2_RXD[2]/GPIO[39]	1	I/O	6.7.4/6-62
GTM2_TGATE[4]	Timer gate 4	Global timers	FEC2_TX_EN/GPIO[42]	1	I/O	6.7.4/6-62
GTM2_TIN[1]	Timer in 1	Global timers	FEC2_COL/GPIO[32]	1	I/O	6.7.4/6-62
GTM2_TIN[2]	Timer in 2	Global timers	FEC2_RX_DV/GPIO[35]	1	I/O	6.7.4/6-62
GTM2_TIN[3]	Timer in 3	Global timers	FEC2_RXD[1]/GPIO[38]	1	I/O	6.7.4/6-62
GTM2_TIN[4]	Timer in 4	Global timers	FEC1_TX_CLK(CLK10)/GPIO[41]	1	I/O	6.7.4/6-62
GTM2_TOUT[1]	Timer out 1	Global timers	FEC2_TXD[0]/GPIO[44]	1	I/O	6.7.4/6-62
GTM2_TOUT[2]	Timer out 2	Global timers	FEC2_TXD[1]/GPIO[45]	1	I/O	6.7.4/6-62
GTM2_TOUT[3]	Timer out 3	Global timers	FEC2_TXD[2]/GPIO[46]	1	I/O	6.7.4/6-62
GTM2_TOUT[4]	Timer out 4	Global timers	FEC2_TX_ER/GPIO[43]	1	I/O	6.7.4/6-62
GPIO[0]	General purpose input/output signal 0	GPIO	SD_CLK/MSRCID[0]	1	I/O	21.2/21-2
GPIO[1]	General purpose input/output signal 1	GPIO	SD_CMD/MSRCID[1]	1	I/O	21.2/21-2
GPIO[2]	General purpose input/output signal 2	GPIO	SD_CD/MSRCID[2]	1	I/O	21.2/21-2
GPIO[3]	General purpose input/output signal 3	GPIO	SD_WP/MSRCID[3]	1	I/O	21.2/21-2
GPIO[4]	General purpose input/output signal 4	GPIO	SD_DAT[0]/MSRCID[4]	1	I/O	21.2/21-2
GPIO[5]	General purpose input/output signal 5	GPIO	SD_DAT[1]/MDVAL	1	I/O	21.2/21-2
GPIO[6]	General purpose input/output signal 6	GPIO	SD_DAT[2]/QE_EXT_REQ_3	1	I/O	21.2/21-2
GPIO[7]	General purpose input/output signal 7	GPIO	SD_DAT[3]/QE_EXT_REQ_1	1	I/O	21.2/21-2

Table 3-1. MPC8309 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/ Page
GPIO[8]	General purpose input/output signal 7	GPIO	RXCAN1/LSRCID0/ $\overline{\text{LCS4}}$	1	I/O	21.2/21-2
GPIO[9]	General purpose input/output signal 0	GPIO	TXCAN1/LSRCID1/ $\overline{\text{LCS5}}$	1	I/O	21.2/21-2
GPIO[10]	General purpose input/output signal 1	GPIO	RXCAN2/LSRCID2/ $\overline{\text{LCS6}}$	1	I/O	21.2/21-2
GPIO[11]	General purpose input/output signal 2	GPIO	TXCAN2/LSRCID3/ $\overline{\text{LCS7}}$	1	I/O	21.2/21-2
GPIO[12]	General purpose input/output signal 3	GPIO	RXCAN3/LSRCID4/LCLK1	1	I/O	21.2/21-2
GPIO[13]	General purpose input/output signal 4	GPIO	TXCAN3/LDVAL	1	I/O	21.2/21-2
GPIO[14]	General purpose input/output signal 5	GPIO	RXCAN4	1	I/O	21.2/21-2
GPIO[15]	General purpose input/output signal 6	GPIO	TXCAN4	1	I/O	21.2/21-2
GPIO[16]	General purpose input/output signal 16	GPIO	FEC1_COL/ GTM1_TIN[1]	1	I/O	21.2/21-2
GPIO[17]	General purpose input/output signal 17	GPIO	FEC1_CRS/ $\overline{\text{GTM1_TGATE}}[1]$	1	I/O	21.2/21-2
GPIO[18]	General purpose input/output signal 18	GPIO	FEC1_RX_CLK(CLK9)	1	I/O	21.2/21-2
GPIO[19]	General purpose input/output signal 19	GPIO	FEC1_RX_DV/ GTM1_TIN[2]	1	I/O	21.2/21-2
GPIO[20]	General purpose input/output signal 20	GPIO	FEC1_RX_ER/ $\overline{\text{GTM1_TGATE}}[2]$	1	I/O	21.2/21-2
GPIO[21]	General purpose input/output signal 21	GPIO	FEC1_RXD[0]	1	I/O	21.2/21-2
GPIO[22]	General purpose input/output signal 22	GPIO	FEC1_RXD[1]/ GTM1_TIN[3]	1	I/O	21.2/21-2
GPIO[23]	General purpose input/output signal 23	GPIO	FEC1_RXD[2]/ $\overline{\text{GTM1_TGATE}}[3]$	1	I/O	21.2/21-2
GPIO[24]	General purpose input/output signal 24	GPIO	FEC1_RXD[3]	1	I/O	21.2/21-2
GPIO[25]	General purpose input/output signal 25	GPIO	FEC1_TX_CLK(CLK10) /GTM1_TIN[4]	1	I/O	21.2/21-2
GPIO[26]	General purpose input/output signal 26	GPIO	FEC_TX_EN/ $\overline{\text{GTM1_TGATE}}[4]$	1	I/O	21.2/21-2
GPIO[27]	General purpose input/output signal 27	GPIO	FEC_TX_ER/ $\overline{\text{GTM1_TOUT}}[4]$	1	I/O	21.2/21-2

Table 3-1. MPC8309 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/ Page
GPIO[28]	General purpose input/output signal 28	GPIO	FEC_TXD[0]/ GTM1_TOUT[1]	1	I/O	21.2/21-2
GPIO[29]	General purpose input/output signal 29	GPIO	FEC_TXD[1]/ GTM1_TOUT[2]	1	I/O	21.2/21-2
GPIO[30]	General purpose input/output signal 30	GPIO	FEC_TXD[2]/ GTM1_TOUT[3]	1	I/O	21.2/21-2
GPIO[31]	General purpose input/output signal 31	GPIO	FEC_TXD[3]	1	I/O	21.2/21-2
GPIO[32]	General purpose input/output signal 32	GPIO	FEC2_COL/GTM2_TIN[1]	1	I/O	21.2/21-2
GPIO[33]	General purpose input/output signal 33	GPIO	FEC2_CRS/ GTM2_TGATE[1]	1	I/O	21.2/21-2
GPIO[34]	General purpose input/output signal 34	GPIO	FEC2_RX_CLK(CLK7)	1	I/O	21.2/21-2
GPIO[35]	General purpose input/output signal 35	GPIO	FEC2_RX_DV/ GTM2_TIN[2]	1	I/O	21.2/21-2
GPIO[36]	General purpose input/output signal 36	GPIO	FEC2_RX_ER/GTM2_TG ATE[2]	1	I/O	21.2/21-2
GPIO[37]	General purpose input/output signal 37	GPIO	FEC2_RXD[0]	1	I/O	21.2/21-2
GPIO[38]	General purpose input/output signal 38	GPIO	FEC2_RXD[1]/GTM2_TIN[3]	1	I/O	21.2/21-2
GPIO[39]	General purpose input/output signal 39	GPIO	FEC2_RXD[2]/GTM2_TG ATE[3]	1	I/O	21.2/21-2
GPIO[40]	General purpose input/output signal 40	GPIO	FEC2_RXD[3]	1	I/O	21.2/21-2
GPIO[41]	General purpose input/output signal 41	GPIO	FEC2_TX_CLK(CLK8) /GTM2_TIN[4]	1	I/O	21.2/21-2
GPIO[42]	General purpose input/output signal 42	GPIO	FEC2_TX_EN/GTM2_TG ATE[4]	1	I/O	21.2/21-2
GPIO[43]	General purpose input/output signal 43	GPIO	FEC2_TX_ER/GTM2_TO UT[4]	1	I/O	21.2/21-2
GPIO[44]	General purpose input/output signal 44	GPIO	FEC2_TXD[0]/GTM2_TO UT[1]	1	I/O	21.2/21-2
GPIO[45]	General purpose input/output signal 45	GPIO	FEC2_TXD[1]/GTM2_TO UT[2]	1	I/O	21.2/21-2
GPIO[46]	General purpose input/output signal 46	GPIO	FEC2_TXD[2]/GTM2_TO UT[3]	1	I/O	21.2/21-2
GPIO[47]	General purpose input/output signal 47	GPIO	FEC2_TXD[3]	1	I/O	21.2/21-2

Table 3-1. MPC8309 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/ Page
GPIO[48]	General purpose input/output signal 48	GPIO	FEC3_COL	1	I/O	21.2/21-2
GPIO[49]	General purpose input/output signal 49	GPIO	FEC3_CR_S	1	I/O	21.2/21-2
GPIO[50]	General purpose input/output signal 50	GPIO	FEC3_RX_CLK(CLK11)	1	I/O	21.2/21-2
GPIO[51]	General purpose input/output signal 51	GPIO	FEC3_RX_DV	1	I/O	21.2/21-2
GPIO[52]	General purpose input/output signal 52	GPIO	FEC3_RX_ER	1	I/O	21.2/21-2
GPIO[53]	General purpose input/output signal 53	GPIO	FEC3_RXD[0]	1	I/O	21.2/21-2
GPIO[54]	General purpose input/output signal 54	GPIO	FEC3_RXD[1]	1	I/O	21.2/21-2
GPIO[55]	General purpose input/output signal 55	GPIO	FEC3_RXD[2]/ TSEC_TMR_TRIG[1]	1	I/O	21.2/21-2
GPIO[56]	General purpose input/output signal 56	GPIO	FEC3_RXD[3]/ TSEC_TMR_TRIG[2]	1	I/O	21.2/21-2
GPIO[57]	General purpose input/output signal 57	GPIO	FEC3_TX_CLK(CLK12) / TSEC_TMR_CLK	1	I/O	21.2/21-2
GPIO[58]	General purpose input/output signal 58	GPIO	FEC3_TX_EN/ TSEC_TMR_GCLK	1	I/O	21.2/21-2
GPIO[59]	General purpose input/output signal 59	GPIO	FEC3_TX_ER/ TSEC_TMR_PP[1]	1	I/O	21.2/21-2
GPIO[60]	General purpose input/output signal 60	GPIO	FEC3_TXD[0]/ TSEC_TMR_PP[2]	1	I/O	21.2/21-2
GPIO[61]	General purpose input/output signal 61	GPIO	FEC3_TXD[1]/ TSEC_TMR_PP[3]	1	I/O	21.2/21-2
GPIO[62]	General purpose input/output signal 62	GPIO	FEC3_TXD[2]/ TSEC_TMR_ALARM[1]	1	I/O	21.2/21-2
GPIO[63]	General purpose input/output signal 63	GPIO	FEC3_TXD[3]/ TSEC_TMR_ALARM[2]	1	I/O	21.2/21-2
$\overline{\text{HDLC1_CD}}$	HDLC1 carrier detect	HDLC	GPIO[4]/TDM1_TFS	1	I/O	21.2/21-2
$\overline{\text{HDLC1_CTS}}$	HDLC1 clear to send	HDLC	GPIO[5]/TDM1_RFS	1	I/O	6.3.2/6-17
$\overline{\text{HDLC1_RTS}}$	HDLC1 ready to send	HDLC	GPIO[6]/ $\overline{\text{TDM1_STROBE}}$ / CFG_RESET_ SOURCE[1]	1	I/O	6.3.2/6-17

Table 3-1. MPC8309 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/ Page
HDLC1_RXCLK (CLK15)	HDLC1 receive clock	HDLC	TDM1_RCK(CLK3) /GPIO[1]	1	I/O	6.3.2/6-17
HDLC1_RXD	HDLC1 receive data	HDLC	TDM1_RD/GPIO[3]	1	I/O	6.3.2/6-17
HDLC1_TXCLK CLK(16)	HDLC1 transmit clock	HDLC	GPIO[0]/QE_BRG[5]/ TDM1_TCK(CLK4)	1	I/O	6.3.2/6-17
HDLC1_TXD	HDLC1 transmit data	HDLC	GPIO[2]/TDM1_TD/ CFG_RESET_SOURCE[0]	1	I/O	6.3.2/6-17
$\overline{\text{HDLC2_CD}}$	HDLC2 carrier detect	HDLC	GPIO[20]/TDM2_TFS	1	I/O	6.3.2/6-17
$\overline{\text{HDLC2_CTS}}$	HDLC2 clear to send	HDLC	GPIO[21]/TDM2_RFS	1	I/O	6.3.2/6-17
$\overline{\text{HDLC2_RTS}}$	HDLC2 ready to send	HDLC	GPIO[22]/ TDM2_STROBE/ CFG_RESET_SOURCE[3]	1	I/O	6.3.2/6-17
HDLC2_RXCLK (CLK13)	HDLC2 receive clock	HDLC	GPIO[17] /TDM2_RCK(CLK5) /QE_BRG[8]	1	I/O	6.3.2/6-17
HDLC2_RXD	HDLC2 receive data	HDLC	GPIO[19]/TDM2_RD	1	I/O	6.3.2/6-17
HDLC2_TXCLK (CLK14)	HDLC2 transmit clock	HDLC	GPIO[16]/QE_BRG[7]/ TDM2_TCK(CLK6)	1	I/O	6.3.2/6-17
HDLC2_TXD	HDLC2 transmit data	HDLC	GPIO[18]/TDM2_TD/ CFG_RESET_SOURCE[2]	1	I/O	6.3.2/6-17
IIC_SCL1	I ² C 1 serial clock	I ² C1	—	1	I/O	17.2/17-3
IIC_SDA1	I ² C 1 serial data	I ² C1	—	1	I/O	17.2/17-3
IIC_SCL2	I ² C 2 serial clock	I ² C2	$\overline{\text{CKSTOP_IN}}$	1	I/O	17.2/17-3
IIC_SDA2	I ² C 2 serial data	I ² C2	$\overline{\text{CKSTOP_OUT}}$	1	I/O	17.2/17-3
$\overline{\text{QUIESCE}}$	Quiescent state	Misc	—	1	O	—
THERM0	Thermal resistor access 0	Misc	—	1	I	—
QE_BRG[1]	QE baud rate generated clock 1	QUICC Engine	USBDR_TXDRXD[2]/ GPIO[34]	1	I/O	6.3.2/6-17
QE_BRG[2]	QE baud rate generated clock 2	QUICC Engine	USBDR_TXDRXD[3]/ GPIO[35]	1	I/O	6.3.2/6-17
QE_BRG[3]	QE baud rate generated clock 3	QUICC Engine	USBDR_TXDRXD[4]/ GPIO[36]	1	I/O	6.3.2/6-17
QE_BRG[4]	QE baud rate generated clock 4	QUICC Engine	USBDR_TXDRXD[5]/ GPIO[37]	1	I/O	6.3.2/6-17

Table 3-1. MPC8309 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/ Page
QE_BRG[5]	QE baud rate generated clock 5	QUICC Engine	HDLC1_TXCLK(CLK16)/GPIO[0]/TDM1_TCK(CLK4)	1	I/O	6.3.2/6-17
QE_BRG[7]	QE baud rate generated clock 7	QUICC Engine	HDLC2_TXCLK(CLK14)/GPIO[16]/TDM2_TCK(CLK6)	1	I/O	6.3.2/6-17
QE_BRG[8]	QE baud rate generated clock 8	QUICC Engine	HDLC2_RXCLK(CLK13)/GPIO[17]/TDM2_RCK(CLK5)	1	I/O	6.3.2/6-17
QE_BRG[9]	QE baud rate generated clock 9	QUICC Engine	USBDR_TXDRXD[6]/GPIO[38]	1	I/O	6.3.2/6-17
QE_BRG[11]	QE baud rate generated clock 11	QUICC Engine	USBDR_TXDRXD[7]/GPIO[39]	1	I/O	6.3.2/6-17
QE_EXT_REQ_1	QE external request 1	QUICC Engine	GPIO[7]/SD_DAT[3]	1	I/O	6.3.2/6-17
QE_EXT_REQ_2	QE external request 2	QUICC Engine	USBDR_STP	1	I/O	6.3.2/6-17
QE_EXT_REQ_3	QE external request 3	QUICC Engine	GPIO[6]/SD_DAT[2]	1	I/O	6.3.2/6-17
QE_EXT_REQ_4	QE external request 4	QUICC Engine	USBDR_NXT/UART2_SIN[1]	1	I/O	6.3.2/6-17
CFG_RESET_SOURCE[0]	Reset configuration source selection 0	Reset and clock	HDLC1_TXD/GPIO[2]/TDM1_TD	1	I/O	4.1.1/4-1
CFG_RESET_SOURCE[1]	Reset configuration source selection 1	Reset and clock	HDLC1_RTS/GPIO[6]/TDM1_STROBE	1	I/O	4.1.1/4-1
CFG_RESET_SOURCE[2]	Reset configuration source selection 2	Reset and clock	HDLC2_TXD/GPIO[18]/TDM2_TD/	1	I/O	4.1.1/4-1
CFG_RESET_SOURCE[3]	Reset configuration source selection 3	Reset and clock	HDLC2_RTS/GPIO[22]/TDM2_STROBE	1	I/O	4.1.1/4-1
CKSTOP_IN	Checkstop in	Reset and clock	IIC_SCL2	1	I/O	4.1.1/4-1
$\overline{\text{CKSTOP_IN}}$	Checkstop in	Reset and clock	$\overline{\text{IRQ}}[2]$	1	I/O	4.1.1/4-1
CKSTOP_OUT	Checkstop out	Reset and clock	IIC_SDA2	1	I/O	4.1.1/4-1
$\overline{\text{CKSTOP_OUT}}$	Checkstop out	Reset and clock	$\overline{\text{IRQ}}[3]$	1	I/O	4.1.1/4-1
$\overline{\text{HRESET}}$	Hard reset	System control	—	1	I/O	—

Table 3-1. MPC8309 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/ Page
PORESET	Power on reset	System control	—	1	I	—
TDM1_TCK (CLK4)	TDM1 transmit clock	TDM	HDLC1_TXCLK(CLK16)/GPIO[0]/QE_BRG[5]	1	I/O	6.3.2/6-17
TDM1_RCK (CLK3)	TDM1 receive clock	TDM	HDLC1_RXCLK(CLK15)/GPIO[1]	1	I/O	6.3.2/6-17
TDM1_TD	TDM1 transmit data	TDM	HDLC1_TXD/GPIO[2]/CFG_RESET_SOURCE[0]	1	I/O	6.3.2/6-17
TDM1_RD	TDM1 receive data	TDM	HDLC1_RXD/GPIO[3]	1	I/O	6.3.2/6-17
TDM1_TFS	TDM1 transmit frame sync	TDM	HDLC1_CD/GPIO[4]	1	I/O	6.3.2/6-17
TDM1_RFS	TDM1 receive frame sync	TDM	HDLC1_CTS/GPIO[5]	1	I/O	6.3.2/6-17
TDM1_STROBE	TDM1 strobe output	TDM	HDLC1_RTS/GPIO[6]/CFG_RESET_SOURCE[1]	1	I/O	6.3.2/6-17
TDM2_RCK (CLK5)	TDM2 receive clock	TDM	HDLC2_RXCLK(CLK13)/GPIO[17]/QE_BRG[8]	1	I/O	6.3.2/6-17
TDM2_RD	TDM2 receive data	TDM	HDLC2_RXD/GPIO[19]	1	I/O	6.3.2/6-17
TDM2_RFS	TDM2 receive frame sync	TDM	HDLC2_CTS/GPIO[21]	1	I/O	6.3.2/6-17
TDM2_STROBE	TDM2 strobe output	TDM	HDLC2_RTS/GPIO[22]/CFG_RESET_SOURCE[3]	1	I/O	6.3.2/6-17
TDM2_TCK (CLK6)	TDM2 transmit clock	TDM	HDLC2_TXCLK(CLK14)/GPIO[16]/QE_BRG[7]	1	I/O	6.3.2/6-17
TDM2_TD	TDM2 transmit data	TDM	HDLC2_TXD/GPIO[18]/CFG_RESET_SOURCE[2]	1	I/O	6.3.2/6-17
TDM2_TFS	TDM2 transmit frame sync	TDM	HDLC2_CD/GPIO[20]	1	I/O	6.3.2/6-17
TEST_MODE	Internal test signal Note: "This pin must always be tied to VSS"	Test	—	1	I	—
TSEC_TMR_TRIG[1]	1588 trigger in 1	IEEE 1588	FEC3_RXD[2]/GPIO[55]	1	I/O	6.3.2/6-17
TSEC_TMR_TRIG[2]	1588 trigger in 2	IEEE 1588	FEC3_RXD[3]/GPIO[56]	1	I/O	6.3.2/6-17
TSEC_TMR_CLK	1588 clock in	IEEE 1588	FEC3_TX_CLK(CLK12)/GPIO[57]	1	I/O	6.3.2/6-17
TSEC_TMR_GCLK	1588 clock out	IEEE 1588	FEC3_TX_EN/GPIO[58]	1	I/O	6.3.2/6-17
TSEC_TMR_PP[1]	1588 timer pulse out 1	IEEE 1588	FEC3_TX_ER/GPIO[59]	1	I/O	6.3.2/6-17
TSEC_TMR_PP[2]	1588 timer pulse out 2	IEEE 1588	FEC3_TXD[0]/GPIO[60]	1	I/O	6.3.2/6-17

Table 3-1. MPC8309 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/ Page
TSEC_TMR_PP[3]	1588 timer pulse out 3	IEEE 1588	FEC3_TXD[1]/GPIO[61]	1	I/O	6.3.2/6-17
TSEC_TMR_ALARM[1]	1588 timer alarm out 1	IEEE 1588	FEC3_TXD[2]/GPIO[62]	1	I/O	6.3.2/6-17
TSEC_TMR_ALARM[2]	1588 timer alarm out 2	IEEE 1588	FEC3_TXD[3]/GPIO[63]	1	I/O	6.3.2/6-17
USBDR_CLK	Clocking signal for ULPI PHY interface	USB	UART2_SIN[2]/ UART2_CTS[1]	1	I	16.2/16-2
USBDR_DIR	Direction of data bus	USB	—	1	I/O	16.2/16-2
USBDR_NXT	Next data	USB	UART2_SIN[1]/QE_EXT_REQ_4	1	I/O	16.2/16-2
USBDR_PCTL[0]	Port control 0	USB	UART2_SOUT[1]/ LB_POR_CFG_BOOT_ECC	1	I/O	16.2/16-2
USBDR_PCTL[1]	Port control 1	USB	UART2_SOUT[2]/ UART2_RTS[1]/ LB_POR_BOOT_ERR	1	O	16.2/16-2
USBDR_PWR_FAULT	USB VBus power fault	USB	—	1	I/O	16.2/16-2
USBDR_STP	End of a transfer on the bus	USB	QE_EXT_REQ_2	1	I/O	16.2/16-2
USBDR_TXDRXD[0]	Data bit 0	USB	GPIO[32]	1	I/O	16.2/16-2
USBDR_TXDRXD[1]	Data bit 1	USB	GPIO[33]	1	I/O	16.2/16-2
USBDR_TXDRXD[2]	Data bit 2	USB	GPIO[33]/QE_BRG[1]	1	I/O	16.2/16-2
USBDR_TXDRXD[3]	Data bit 3	USB	GPIO[35]/QE_BRG[2]	1	I/O	16.2/16-2
USBDR_TXDRXD[4]	Data bit 4	USB	GPIO[36]/QE_BRG[3]	1	I/O	16.2/16-2
USBDR_TXDRXD[5]	Data bit 5	USB	GPIO[37]/QE_BRG[4]	1	I/O	16.2/16-2
USBDR_TXDRXD[6]	Data bit 6	USB	GPIO[38]/QE_BRG[9]	1	I/O	16.2/16-2
USBDR_TXDRXD[7]	Data bit 7	USB	GPIO[39]/QE_BRG[11]	1	I/O	16.2/16-2
MSRCID[0]	DDR memory debug source ID 0	DDR	GPIO[0]/SD_CLK	1	I/O	10.3/10-3
MSRCID[1]	DDR memory debug source ID 1	DDR	GPIO[1]/SD_CMD	1	I/O	10.3/10-3
MSRCID[2]	DDR memory debug source ID 2	DDR	GPIO[2]/SD_CD	1	I/O	10.3/10-3
MSRCID[3]	DDR memory debug source ID 3	DDR	GPIO[3]/SD_WP	1	I/O	10.3/10-3
MSRCID[4]	DDR memory debug source ID 4	DDR	GPIO[4]/SD_DAT[0]	1	I/O	10.3/10-3
MDVAL	Memory debug data valid	DDR	GPIO[5]/SD_DAT[1]	1	I/O	10.3/10-3

Table 3-2 lists the signals in alphabetical order.

3.2 Output Signal States During Reset

When a system reset is recognized, the device aborts all current internal and external transactions (with the exception of RTC) and releases all bidirectional I/O signals to a high-impedance state. See Chapter 4, “Reset, Clocking, and Initialization,” for a complete description of the reset functionality.

During reset, the device ignores most input signals (except for the reset configuration signals) and drives most of the output-only signals to an inactive state. The bidirectional pins are held input state and the input values are ignored. Table 3-2 shows the states of the output-only signals.

Table 3-2. Output Signal States During System Reset

Interface	Signal	State During Reset
MA[0:13]	DDR address	High-Z
MECC[0:7]	DDR ECC bits	High-Z
MBA[0:2]	DDR bank select	High-Z
\overline{MCAS}	DDR column address strobe	High-Z
MCK[0:1]	DDR differential clocks	High-Z
\overline{MCK} [0:1]	DDR differential clocks	High-Z
MCKE	DDR clock enable	Driven Low
\overline{MCS} [0:1]	DDR chip select	Driven Low
MDM[0:3]	DDR data mask	Driven Low
MDM[8]	DDR data mask	Driven Low
MDQ[0:31]	DDR data	Driven Low
MDQS[0:3]	DDR data strobe	Driven Low
MDQS[8]	DDR data strobe	Driven Low
MODT[0:1]	DDR on-die termination	Driven Low
\overline{MRAS}	DDR row address strobe	Driven Low
MVREF	DDR DRAM reference	Driven Low
\overline{MWE}	DDR write enable	Driven Low
LA[16:25]	LBC port address	Active—used to load reset configuration word
LAD[0:15]	LBC address data	Active—used to load reset configuration word
LALE	LBC address latch enable	Active—used to load reset configuration word
LCLK0	LBC clock 0	High-Z

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

Interface	Signal	State During Reset
$\overline{\text{LCS0}}$	LBC chip select 0–3	Active—used to load reset configuration word
$\overline{\text{LCS}}[1:3]$	LBC chip select 0–3	High
$\overline{\text{LCS}}[4]$	LBC chip select 4	
LGPL[0]/LFCLE	LBC UPM general purpose line 0	High-Z
UART1_SOUT[1:2]	DUART serial data out	Driven High
FEC_MDC	Ethernet management data clock	Driven Low
TDO	Test data out	High-Z
$\overline{\text{QUIESCE}}$	Quiescent state	High
USBDR_STP	End of a transfer on the bus	High

NOTE

All the GPIOs have PULL-UP enabled on reset. It can be changed later by setting configuration bits in the register GPR_1 (General Purpose Register 1). Refer to [Chapter 6, “System Configuration”](#).



Chapter 4

Reset, Clocking, and Initialization

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

4.1 External Signals

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

4.1.1 Reset Signals

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

Table 4-1. System Control Signals

Signal	I/O	Description
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 For timing information, see <i>MPC8309 PowerQUICC II Pro Integrated Communications Processor Hardware Specification</i> .
		Reset State Always input.
HRESET	I/O	Hard reset. Causes the device to abort all current internal and external transactions and set most registers to their default values. HRESET can be asserted completely asynchronously with respect to all other signals. The device can detect an external assertion of HRESET while the device is not asserting hard reset. HRESET is an open-drain signal.
		State Meaning Asserted—An external agent or internal hardware has triggered a hard reset. The internal hardware drives 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 SYS_CLK_IN 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. Pull-up is enabled by default (out of reset) for these pins. It can be changed at a later time by configuring General Purpose Register 1.

Table 4-1. System Control Signals (continued)

Signal	I/O	Description	
CFG_RESET_SOURCE[0:3]	I	Reset configuration word source selection. These signals are on device pins that have other functions when the device is not in reset. They are sampled during the assertion of $\overline{\text{PORESET}}$ to determine the interface from which the device loads the reset configuration words.	
		State Meaning	See Section 4.3.1.1, “Reset Configuration Word Source.”
		Timing	These signals are sampled during the assertion of $\overline{\text{PORESET}}$ after a stable clock is supplied ($\overline{\text{PORESET}}$ flow) and must be pulled high or low by external resistors as long as $\overline{\text{HRESET}}$ is asserted.
		Requirements	During $\overline{\text{PORESET}}$ and $\overline{\text{HRESET}}$ flows, all other signal drivers connected to these signals must be in the high-impedance state. For proper resistor values to pull reset configuration signals high or low, see <i>MPC8309 PowerQUICC II Pro Integrated Communications Processor Hardware Specification</i> .
	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 4-7 for the internal distribution of clocks in the device.

Table 4-2. External Clock Signals

Signal	I/O	Description	
SYS_CLK_IN	I	System clock. In PCI host mode, SYS_CLK_IN is the primary input clock. SYS_CLK_IN directly feeds the PCI output clock dividers and is driven out on the PCI_SYNC_OUT signal for de-skewing external PCI clocks routing. If the device is used as PCI agent, PCI_CLK can be provided from a PCI source. In this case, SYS_CLK_IN may no longer be needed and should be tied low. If SYS_CLK_IN is the clock source, SYS_XTAL_IN should be tied low and SYS_XTAL_OUT should be left unconnected.	
		Timing	Assertion/Negation—For timing information, see <i>MPC8309 PowerQUICC II Pro Integrated Communications Processor Hardware Specification</i> .
		Requirements	Should be tied low in PCI agent mode.
		Reset State	Always input.
SYS_XTAL_IN		Crystal input. SYS_XTAL_IN allows the system clock to be provided using an external crystal oscillator. If a crystal source is used, SYS_CLK_IN should be tied low.	
		Timing	Assertion/Negation—See the <i>MPC8309 PowerQUICC II Pro Integrated Communications Processor Hardware Specification</i> for timing information.
		Reset State	Always input.

Table 4-2. External Clock Signals (continued)

Signal	I/O	Description
SYS_XTAL_OUT		Crystal output. SYS_XTAL_OUT allows the system clock to be provided through an external crystal oscillator. If a crystal source is used, SYS_CLK_IN should be tied low.
	Timing	Assertion/Negation—See the <i>MPC8309 PowerQUICC II Pro Integrated Communications Processor Hardware Specification</i> for timing information
	Reset State	Always output.
PCI_SYNC_IN		PCI input synchronization clock
	Timing	Assertion/Negation—For timing information, see <i>MPC8309 PowerQUICC II Pro Integrated Communications Processor Hardware Specification</i> .
	Reset State	Always input
PCI_SYNC_OUT		PCI output synchronization clock
	Timing	Assertion/Negation—For timing information, see <i>MPC8309 PowerQUICC II Pro Integrated Communications Processor Hardware Specification</i> .
	Reset State	Always output, toggling in PCI host mode.
QE_CLK_IN	I	QE_CLK_IN is the the reference clock for the QE PLL.
	Timing	Assertion/Negation—For timing information, see <i>MPC8309 PowerQUICC II Pro Integrated Communications Processor Hardware Specification</i> .
	Reset State	Always input.
FEC1_RX_CLK, FEC2_RX_CLK, FEC3_RX_CLK	I	Ethernet Receive clock
	Timing	Assertion/Negation—For timing information, see <i>MPC8309 PowerQUICC II Pro Integrated Communications Processor Hardware Specification</i> .
	Reset State	Always input.
FEC1_TX_CLK, FEC2_TX_CLK, FEC3_TX_CLK	I	Ethernet Transmit Clock.
	Timing	Assertion/Negation—For timing information, see <i>MPC8309 PowerQUICC II Pro Integrated Communications Processor Hardware Specification</i> .
	Reset State	Always input.
HDLC1_TCK, HDLC2_TCK, TDM1_TCK, TDM2_TCK	I	HDLC/TDM Transmit clock
	Timing	Assertion/Negation—For timing information, see <i>MPC8309 PowerQUICC II Pro Integrated Communications Processor Hardware Specification</i> .
	Reset State	Always input.
HDLC1_RCK, HDLC2_RCK, TDM1_RCK, TDM2_RCK	I	HDLC/TDM Recieve clock
	Timing	Assertion/Negation—For timing information, see <i>MPC8309 PowerQUICC II Pro Integrated Communications Processor Hardware Specification</i> .
	Reset State	Always input.

Table 4-2. External Clock Signals (continued)

Signal	I/O	Description	
USBDR_CLK	I	USB controller clock	
		Timing	Assertion/Negation—For timing information, see <i>MPC8309 PowerQUICC II Pro Integrated Communications Processor Hardware Specification</i> .
		Reset State	Always input.
RTC_PIT_CLOCK	I	This signal is used as timebase for real time clock module. RTC_PIT_CLK is used as the source for the following modules - RTC, PIT and FlexCAN. For FlexCAN, it is used as the alternative source of CAN clock.	
		Timing	32.768 KHz typical.
		Reset State	Always input.

4.2 Functional Description

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

4.2.1 Reset Operations

The device has the following inputs to the reset logic:

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

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

4.2.1.1 Reset Causes

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

Table 4-3. Reset Causes

Name	Description
Power-on reset ($\overline{\text{PORESET}}$)	Input signal. Asserting this signal initiates the power-on reset flow that resets the entire device except RTC and configures various attributes of the device including its clock modes.
Hard reset ($\overline{\text{HRESET}}$)	A bidirectional I/O signal. The device can detect an external assertion of $\overline{\text{HRESET}}$ only while it is not asserting hard reset. $\overline{\text{HRESET}}$ is an open-drain signal.

Table 4-3. Reset Causes (continued)

Name	Description
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.
Software hard reset	A hard reset sequence can be initialized by writing to a memory-mapped register (RCR).

4.2.1.2 Reset Actions

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

- Power-on reset has the greatest impact, resetting the entire device, including clock logic and error capture registers.
- Hard reset resets the entire device, excluding RTC module, clock logic, and error capture registers.

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
Resets: PLLs, clocks, and error capture registers	Yes	No
Resets: DDR controller, LBC, I/O multiplexors, GTM, PIT, GPIO, system configuration, and local access windows	Yes	Yes
Resets other internal logic	Yes	Yes
Reset configuration words loaded	Yes	Yes
\overline{HRESET} driven by SoC	Yes	Yes
Hard reset to e300 core	Yes	Yes
\overline{SRESET} : High priority interrupt to the e300 core	No	No

4.2.2 Power-On Reset Flow

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

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

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

1. Power is applied to meet the specifications in the device data sheet.
2. The system asserts $\overline{\text{PORESET}}$ and $\overline{\text{TRST}}$, causing all registers to be initialized to their default states and most I/O drivers to be released to high-impedance.
3. The system applies a stable SYS_CLK_IN signal and stable reset configuration inputs (CFG_RESET_SOURCE).
4. The system negates $\overline{\text{PORESET}}$ after at least 32 stable SYS_CLK_IN clock cycles.
5. The device samples the reset configuration input signals to determine the reset configuration words source.
6. The device starts loading the reset configuration words.
Loading time depends on the reset configuration word source.
7. When the reset configuration word low is loaded, the system PLL begins to lock.
When the system PLL is locked, csb_clk is supplied to the core PLL.
8. The core PLL begins to lock.
9. The device drives $\overline{\text{HRESET}}$ asserted until the e300 PLL is locked and the reset configuration words are loaded.
10. The user optionally negates $\overline{\text{HRESET}}$ if it was not negated earlier.
JTAG logic must always be initialized by asserting $\overline{\text{TRST}}$. If the JTAG signals are not used, $\overline{\text{TRST}}$ should be connected directly to $\overline{\text{PORESET}}$. $\overline{\text{TRST}}$ must not remain asserted after the negation of $\overline{\text{PORESET}}$.
11. The internal reset to the core and the rest of the logic is negated. I/O drivers are enabled.
12. The device stops driving $\overline{\text{HRESET}}$. The reset to the e300 core is negated and the core is enabled. The boot sequencer, if enabled, is released, causing it to load configuration data from serial ROMs, as described in [Section 17.4.5, “Boot Sequencer Mode.”](#)
13. If the e300 core is required to proceed before the boot sequencer finishes, the boot sequencer should enable boot vector fetch by clearing $\text{ACR}[\text{COREDIS}]$ as described in [Section 7.2.1, “Arbiter Configuration Register \(ACR\).”](#)
14. The boot vector fetch by the core can proceed, if enabled.

The device is now in its ready state.

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

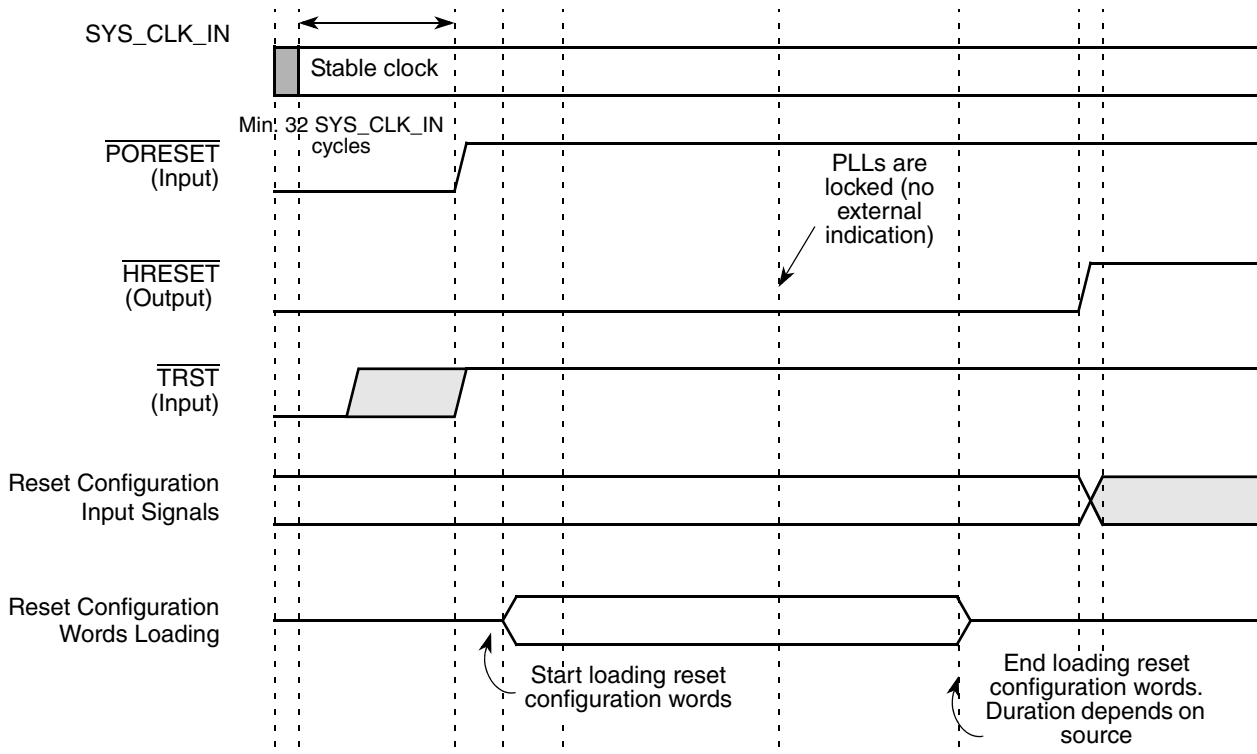


Figure 4-1. Power-On Reset Flow

NOTE

It takes about 150 SYS_CLK_IN cycles between the negation of PORESET and access to reset configuration word.

4.2.3 Hard Reset Flow

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

NOTE

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

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

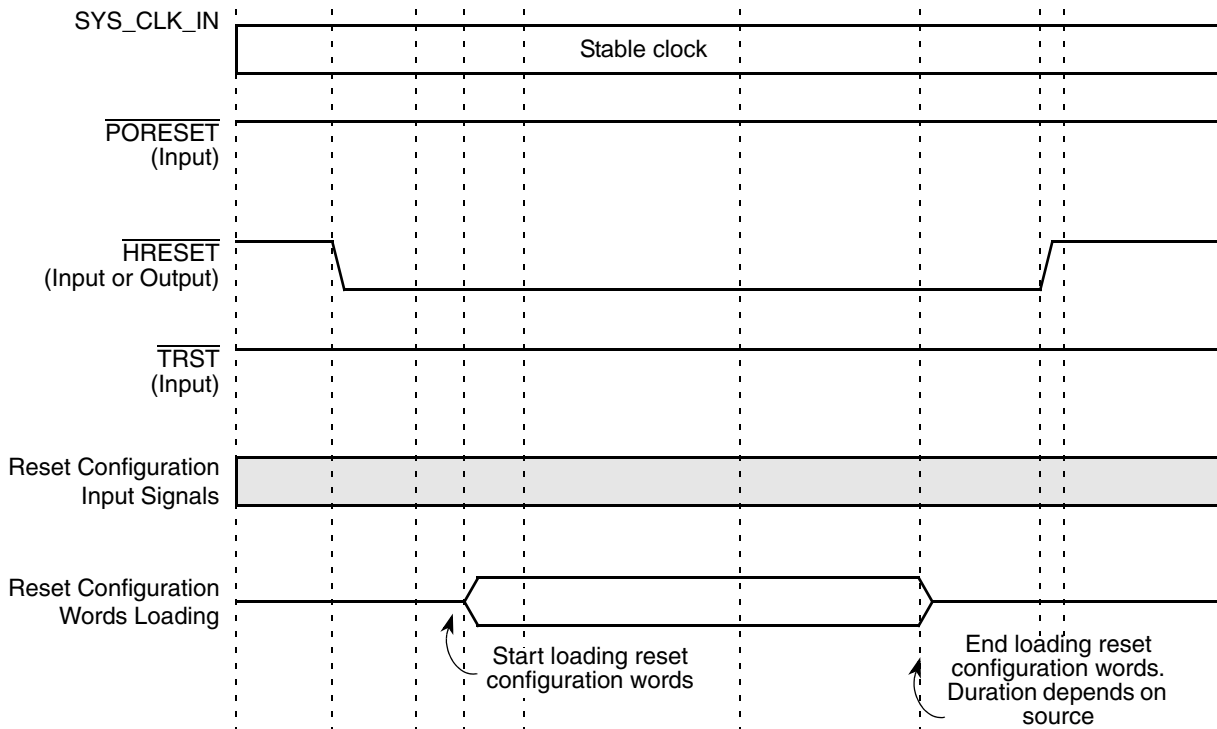


Figure 4-2. Hard Reset Flow

4.3 Reset Configuration

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

4.3.1 Reset Configuration Signals

Reset configuration input signals are on device pins that have other functions when the device is not in reset state. These input signals are sampled into registers during the assertion of `PORESET`, after a stable clock is supplied (`SYS_CLK_IN`), and must be pulled high or low by external resistors as long as `HRESET` is asserted. While the `PORESET` or `HRESET` signal is asserted, all other signal drivers connected to these signals must be in the high-impedance state. For proper resistor values for pulling reset configuration signals high or low, see *MPC8309 PowerQUICC II Pro Integrated Communications Processor Hardware Specification*.

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

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

NOTE

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

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

4.3.1.1 Reset Configuration Word Source

The reset configuration word source options, shown in Table 4-5, select whether the device loads a reset configuration word from NOR Flash, NAND Flash, or an I²C EEPROM or uses hard-coded default options. The value of these signals also affects the duration of power-on and hard reset sequences. In any case, the reset sequence does not exceed 1ms.

Table 4-5. Reset Configuration Words Source

CFG_RESET_SOURCE[0:3]	Meaning
0000	Reset configuration word is loaded from NOR Flash
0001	Reset configuration word is loaded from NAND Flash memory (8-bit small page).
0010	Reserved
0011	Reserved
0100	Reset configuration word is loaded from an I ² C EEPROM. SYS_CLK_IN is in the range of 24–66.666 MHz.
0101	Reset configuration word is loaded from NAND Flash memory (8-bit large page).
0110	Reserved
0111	Reserved
1000	Hard-coded option 0. Reset configuration word is not loaded.
1001	Hard-coded option 1. Reset configuration word is not loaded.
1010	Hard-coded option 2. Reset configuration word is not loaded.
1011	Reset configuration word is loaded from NOR Flash.
1100	Hard-coded option 4. Reset configuration word is not loaded.
1101	Hard-coded option 5. Reset configuration word is not loaded.
1110	Hard-coded option 6. Reset configuration word is not loaded.
1111	Hard-coded option 7. Reset configuration word is not loaded.

4.3.1.2 Selecting Reset Configuration Input Signals

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

Table 4-6. Selecting Reset Configuration Input Signals

I ² C EEPROM Configuration Words	SYS_CLK_IN Frequency	CFG_RESET_SOURCE[0:3]	Reset Sequence Duration in SYS_CLK_IN Cycles	Duration
No	33 MHz	0000 (RCW loaded from NOR Flash)	15210	456 μs
Yes	33 MHz	0100 (I ² C EEPROM)	106534	196 μs

4.3.2 Reset Configuration Words

The reset configuration words control the clock ratios and other basic device functions such as boot location and endian mode. The reset configuration words are loaded from NOR Flash, NAND Flash, or the I²C interfaces or from hard-coded values during the power-on or hard reset flows. See [Section 4.3.1, “Reset Configuration Signals,”](#) for information on the reset configuration word source. Although the configuration reset words are loaded during hard reset flows, the clocks and PLL modes are reset only when $\overline{\text{PORESET}}$ is asserted during a power-on reset flow. See [Section 4.2.1.2, “Reset Actions.”](#) The values of fields in the reset configuration words registers (RCWLR and RCWHR) reflect only their state during the reset flow. Some of these parameters and modes can be modified by changing their values in the memory-mapped registers of other units, which does not affect RCWLR and RCWHR.

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

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

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

4.3.2.1 Reset Configuration Word Low Register (RCWLR)

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

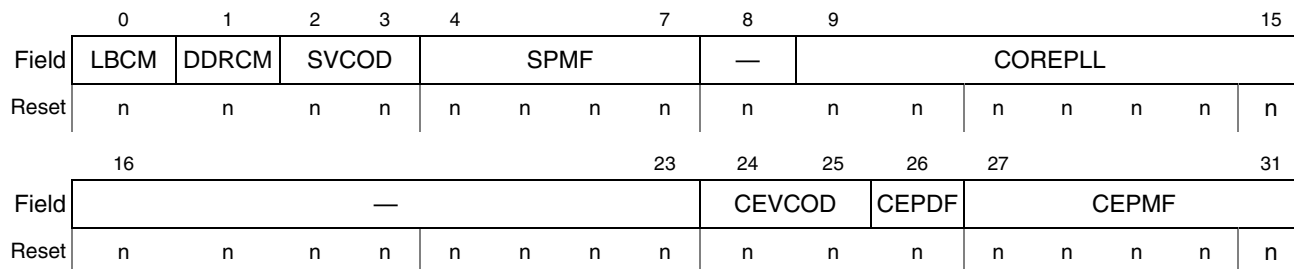


Figure 4-3. Reset Configuration Word Low Register (RCWLR)

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

Table 4-7. RCWLR Bit Settings

Bits	Name	Description
0	LBCM	Local bus memory controller clock mode. Selects the local bus controller clock ratio. The local bus memory controller operates with a frequency equal to the frequency of <i>csb_clk</i> . This bit should be cleared. LBC controller clock: <i>csb_clk</i> 0 1:1
1	DDRCM	DDR SDRAM memory controller clock mode. Selects the DDR SDRAM memory controller clock ratio. The DDR SDRAM memory controller operates at twice the frequency of the <i>csb_clk</i> . 0 Reserved 1 <i>csb_clk</i> ratio is 2:1
2–3	SVCOD	System PLL VCO division. See Section 4.3.2.1.1, “System PLL VCO Division.”
4–7	SPMF	System PLL multiplication factor. See Section 4.3.2.1.1, “System PLL VCO Division,” for more information.
8	—	Reserved, should be cleared
9–15	COREPLL	Core PLL configuration. COREPLL sets the ratio between the e300 core clock and the internal <i>csb_clk</i> of the device. For information on the encodings for COREPLL, see <i>MPC8309 PowerQUICC II Pro Integrated Communications Processor Hardware Specification</i> .
16–23	—	Reserved, should be cleared.
24–25	CEVCOD	QUICC Engine PLL VCO division. Establishes the internal ratio between the QUICC Engine PLL VCO point and the QUICC Engine PLL output. 00 2 01 4 10 8 11 Reserved Notes: Set CEVCOD to 00 (Division factor of 2) for QE frequency below 150 MHz. Set CEVCOD to 01 (Division factor of 4) for QE frequency above 150 MHz.

Table 4-7. RCWLR Bit Settings (continued)

Bits	Name	Description
26	CEPDF	QUICC Engine PLL division factor. Selects whether the PLL output is halved. By setting this bit, non-integer ratios between the primary clock input and <i>qe_clk</i> can be achieved. 0 <i>qe_clk</i> = primary clock input × CEPMF 1 <i>qe_clk</i> = (primary clock input × CEPMF) / 2
27–31	CEPMF	QUICC Engine PLL multiplication factor. See Section 4.3.2.1.3, “QUICC Engine PLL Multiplication Factor,” for more information

4.3.2.1.1 System PLL VCO Division

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

[Table 4-8](#) describes the setting of SVCOD bits.

Table 4-8. System PLL VCO Division

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

NOTE

For the frequency of operation for MPC8309, 00 is the only applicable value of SVCOD.

4.3.2.1.2 System PLL Configuration

The system PLL ratio reset, shown in [Table 4-9](#), establishes the clock ratio between the SYS_CLK_IN signal and the internal *csb_clk* of the device. *csb_clk* drives internal units and feeds the e300 core PLL.

Table 4-9. System PLL Ratio

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

4.3.2.1.3 QUICC Engine PLL Multiplication Factor

The RCWLR field CEPMF (QUICC Engine PLL multiplication factor), shown in [Table 4-10](#), along with the QUICC Engine PLL division factor (CEPDF,) establishes the ratio between *qe_clk* and the primary input clock.

Table 4-10. QUICC Engine PLL Multiplication Factor

RCWLR Bits	Field Name	Value (Binary)	QUICC Engine Block PLL Multiplication Factor
27–31	CEPMF	00000	Reserved, should be cleared.
		00001	Reserved, should be cleared.
		00010	2
		00011	3
		00100	4
		00101	5
		00110	6
		00111	7
		01000	8
		01001–11111	Reserved, should be cleared.

4.3.2.2 Reset Configuration Word High Register (RCWHR)

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

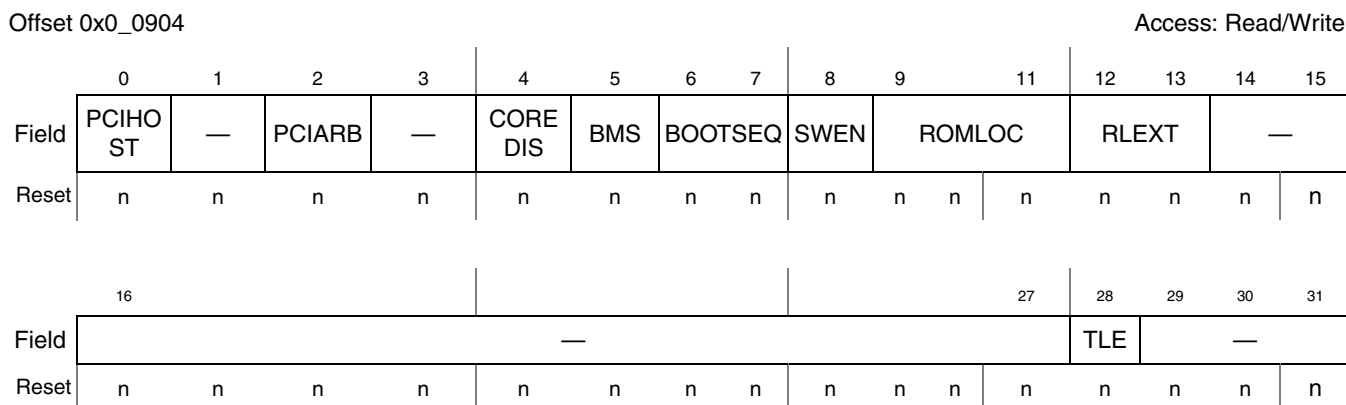


Figure 4-4. Reset Configuration Word High Register (RCWHR)

[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.								
1	—	Reserved, should be cleared.								
2	PCIARB	<p>PCI internal arbiter mode. Enables the on-chip PCI arbiter.</p> <p>0 On-chip PCI arbiter is disabled. External arbitration is required. 1 On-chip PCI arbiter is enabled.</p> <p>The value of PCIARB also defines the function of the PCI arbitration signals that are multiplexed with CompactPCI signals, as follows:</p> <table border="1" style="margin: 10px auto; border-collapse: collapse; text-align: center;"> <thead> <tr> <th style="width: 50%;">Pin Function When PCIARB = 0</th> <th style="width: 50%;">Pin Function When PCIARB = 1</th> </tr> </thead> <tbody> <tr> <td>CPCI_HS_ES</td> <td>$\overline{\text{PCI_REQ}}[1]$</td> </tr> <tr> <td>CPCI_HS_LED</td> <td>$\overline{\text{PCI_GNT}}[1]$</td> </tr> <tr> <td>CPCI_HS_ENUM</td> <td>$\overline{\text{PCI_GNT}}[2]$</td> </tr> </tbody> </table>	Pin Function When PCIARB = 0	Pin Function When PCIARB = 1	CPCI_HS_ES	$\overline{\text{PCI_REQ}}[1]$	CPCI_HS_LED	$\overline{\text{PCI_GNT}}[1]$	CPCI_HS_ENUM	$\overline{\text{PCI_GNT}}[2]$
Pin Function When PCIARB = 0	Pin Function When PCIARB = 1									
CPCI_HS_ES	$\overline{\text{PCI_REQ}}[1]$									
CPCI_HS_LED	$\overline{\text{PCI_GNT}}[1]$									
CPCI_HS_ENUM	$\overline{\text{PCI_GNT}}[2]$									
3	—	Reserved, should be cleared.								
4	COREDIS	<p>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 7.2.1, “Arbiter Configuration Register (ACR)”.</p> <p>This bit must be set when the boot sequencer is enabled to initiate the device (BOOTSEQ is 01 or 10). Otherwise, unpredictable behavior occurs.</p> <p>0 The core can boot without waiting for configuration by an external master. 1 Core boot hold-off mode. The core is prevented from fetching instruction opcode until this bit is cleared.</p>								

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

Bits	Name	Description
5	BMS	Boot memory space. See Section 4.3.2.2.1, “Boot Memory Space (BMS),” for more information.
6–7	BOOTSEQ	Boot sequencer configuration. See Section 4.3.2.2.2, “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. This bit combined with bit RLEXT determines where the device boots from. See Section 4.3.2.2.3, “Boot ROM Location,” for more information.
12–13	RLEXT	Boot ROM location extension. This bit combined with bit ROMLOC determines where the device boots from. See Section 4.3.2.2.3, “Boot ROM Location,” for more information. 00 Legacy mode—allows for booting from on-chip peripherals. For more information, see Table 4-14 . 01 NAND Flash mode—allows for booting from NAND flash devices. For more information, see Table 4-14 . 10 Reserved 11 Reserved
14–27	—	Reserved, should be cleared.
28	TLE	True little-endian. See Section 4.3.2.2.4, “e300 Core True Little-Endian,” for more information.
29–31	—	Reserved, should be cleared.

4.3.2.2.1 Boot Memory Space (BMS)

BMS defines the initial value of the e300 core MSR[IP] bit, which specifies the location of the interrupt vectors (including the hard reset exception vector). The device defines the default boot ROM memory space to be 8 Mbytes at addresses 0x0000_0000 to 0x007F_FFFF or 0xFF80_0000 to 0xFFFF_FFFF. When the core comes out of reset, if it is enabled to boot, it fetches boot code from one of two addresses, 0x0000_0100 or 0xFFFF_0100, and exceptions are vectored to the physical addresses, 0x000n_nnnn or 0xFFFFn_nnnn appropriately. This bit specifies whether an interrupt vector offset is prepended with 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 4-12](#), specifies both the device boot ROM address window and the initial e300 core boot address.

Table 4-12. Boot Memory Space

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

4.3.2.2.2 Boot Sequencer Configuration

The boot sequencer configuration options, shown in [Table 4-13](#), allow the boot sequencer to load configuration data from the serial ROM located on the I²C port before the host tries to configure the device. These options also specify normal or extended I²C addressing modes. See [Section 17.4.5, “Boot Sequencer Mode.”](#)

Table 4-13. Boot Sequencer Configuration

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

NOTE

When the boot sequencer is enabled, the e300 core must be prevented from fetching boot code by setting the core disable reset configuration word field (COREDIS) as described in [Section 4.3.2.2, “Reset Configuration Word High Register \(RCWHR\).”](#) If the e300 core is required to proceed, the boot sequencer should enable boot vector fetch by clearing ACR[COREDIS] as described in [Section 7.2.1, “Arbiter Configuration Register \(ACR\).”](#)

4.3.2.2.3 Boot ROM Location

The device defines the default boot ROM address range to be 8 Mbytes at addresses 0x0000_0000 to 0x007F_FFFF or 0xFF80_0000 to 0xFFFF_FFFF (selected by the BMS reset configuration word field). However, the on-chip peripheral that manages these boot ROM accesses can be selected at power up.

The boot ROM location reset configuration word field, shown in [Table 4-14](#), establishes the location of boot ROM. The exact boot ROM location table to be used is defined by the setting of RCWHR[RLEXT] bits, as shown in [Table 4-11](#). 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 4-14. Boot ROM Location

RCWHR Bits	Field Name	Value (Binary)	Meaning	
			Legacy Mode (RLEXT = 00)	NAND Flash Mode (RLEXT = 01)
9–11	ROMLOC	000	DDR SDRAM	Reserved
		001	PCI	Local bus NAND Flash—8-bit small page ROM
		010	eSDHC Boot ; BMS has to be 1 for eSDHC boot.	Reserved
		011	SPI Boot ; BMS has to be 1 for SPI boot.	Reserved
		100	Reserved	Reserved
		101	Local bus GPCM—8-bit ROM	Local bus NAND Flash—8-bit large page ROM
		110	Local bus GPCM—16-bit ROM	Reserved
		111	Reserved	Reserved

The local access window of the selected boot ROM interface is enabled and initialized with the proper base address and size, as described in [Section 6.2, “Local Memory Map Overview and Example.”](#)

4.3.2.2.4 e300 Core True Little-Endian

The true little-endian reset configuration word field, shown in [Table 4-15](#), selects whether the e300 core operates in big-endian mode or true little-endian mode at reset.

Table 4-15. e300 Core True Little-Endian

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

4.3.3 Loading the Reset Configuration Words

The device loads the reset configuration words from a local bus EEPROM, a local bus NAND Flash, or an I²C serial EEPROM, or uses hard-coded configuration, as selected by the reset configuration inputs

described in [Section 4.3.1, “Reset Configuration Signals.”](#) The following sections describe each of these options.

4.3.3.1 Loading from Local Bus

The reset configuration words are assumed to reside in an EEPROM or NOR Flash or NAND Flash device connected to $\overline{LCS0}$ 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 4-16](#) 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 4-16](#) are always read on byte lane LD[0:7] regardless of the port size.

Table 4-16. Local Bus Configuration EEPROM Addresses

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

[Table 4-17](#) shows the data structure of the local bus device containing the reset configuration words (RCWL and RCWH).

Table 4-17. Local Bus Reset Configuration Words Data Structure

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

4.3.3.1.1 Local Bus Controller Setting

The device uses GPCM to load the reset configuration from EEPROM or NOR Flash. The device reads 64 bytes in this case. The local bus controller's registers setting is set according to [Table 4-18](#).

The device uses FCM to load the reset configuration from NAND Flash. The device reads 512 bytes if small page size NAND Flash is used or 2048 bytes if large page NAND Flash is used. The local bus controller's registers setting are set according to [Table 4-18](#).

Table 4-18. Local Bus Controller Setting when Loading RCW

CFG_RESET_SOURCE	Meaning	BR0[PS]	BR0[MSEL]	OR0[SCY]	OR0[PGS]
0000	NOR Flash	10	000	1111	NA
0001	NAND Flash, 8 bit, small page	01	001	0010	0
0101	NAND Flash, 8 bit, large page	01	001	0010	1

4.3.3.2 Loading from I²C EEPROM

The device is capable of loading the reset configuration word from the I²C interface. If the device is configured to load the reset configuration word from the I²C interface according to the reset configuration input signals, the device 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 more information on I²C interface and the boot sequencer, see [Section 17.4.5, "Boot Sequencer Mode."](#)

NOTE

When reset configuration words are loaded from an I²C EEPROM, 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 0xAA_55AA. 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 4-5](#).

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

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

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

When the I²C operates in reset configuration mode, the cyclic redundancy check (CRC) is ignored, as well as any registers following the first two reset configuration words.

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

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

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

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

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

Figure 4-6. EEPROM Contents

4.3.3.2.4 Reset Configuration Load Fail

Failure of reset configuration load by the I²C boot sequencer can be caused by an incorrect EEPROM data structure or I²C bus problem. If a reset configuration load failure occurs, due to preamble fail or any other I²C bus error detection, the device continuously attempts to reload the hard reset configuration words from the I²C bus. The device does not negate $\overline{\text{HRESET}}$ and remains in hard reset state until the RCWs are successfully loaded or the PORESET flow is restarted.

4.3.3.3 Default Reset Configuration Words

If the device is configured not to load the reset configuration words from NOR Flash, NAND Flash, or an I²C EEPROM, it can also be initialized with one of five hard-coded default options, selected by the reset configuration input signals, CFG_RESET_SOURCE[0:3].

The reset configuration words are driven internally with the values shown in [Table 4-19](#), [Table 4-20](#), and [Table 4-21](#).

Table 4-19. RCW Values Corresponding to Hard Coded Options

CFG_RESET_SOURCE	RCWLR [0:31]	RCWHR [0:31]
0b1000	42040003	A4600000
0b1001	44040007	A4600000
0b1010	42230003	A4600000
0b1100	44050006	A4600000
0b1101	44040003	A4200000
0b1110	44050007	A4200000
0b1111	44050007	A4300000

Table 4-20. Hard Coded Reset Configuration Word Low Fields Values

RCW bits	0	1	2–3	4–7	8	9–15	16–23	24–25	26	27–31
Fields	LBCM	DDRCM	SVCOD	SPMF	Reserve	COREPLL	Reserve	CEVCOD	CEPDF	CEPMF
Meaning	LBC Control clock :	DDR Control ler clock:	System VCO PLL division factor	csb_clk : SYS_C LK_IN		Core Clock: csb_clk		QUICC Engine VCO PLL Division factor	QUICC Engine PLL Divisio n Factor	QUICC Engine PLL Multipli cation Factor
CFG_RESET_SOURCE value	0 1:1	0 1:1 Reserve 1 2:1								
1000	0	1	00	0010	0	0000100	0000000 0	00	0	00011
1001	0	1	00	0100	0	0000100	0000000 0	00	0	00111
1010	0	1	00	0010	0	0100011	0000000 0	00	0	00011

Table 4-20. Hard Coded Reset Configuration Word Low Fields Values (continued)

1100	0	1	00	0100	0	0000101	0000000 0	00	0	00110
1101	0	1	00	0100	0	0000100	0000000 0	00	0	00011
1110	0	1	00	0100	0	0000101	0000000 0	00	0	00111
1111	0	1	00	0100	0	0000101	0000000 0	00	0	00111

Table 4-21 defines the hard-coded reset configuration word high fields values. These values select hard-coded reset configuration words options, as described in Section 4.3.1.1, “Reset Configuration Word Source.”

Table 4-21. Hard-Coded Reset Configuration Word High Field Values

Bits	Name	Field Values when CFG_RESET_SOURCE[0-3] = 1000-1111							Meaning
		1000	1001	1010	1100	1101	1110	1111	
0	PCIHOST	1							—
1	Reserved	0							—
2	PCIARB	1							—
3	Reserved	0							—
4	COREDIS	0							e300 core is enabled (boot hold-off)
5	BMS	1							Boot memory space is 0xFF80_0000–0xFFFF_FFFF. MSR[IP] initial value is 0b1.
6-7	BOOTSEQ	00							Boot sequencer is disabled
8	SWEN	0							Software watchdog disabled
9-11	ROMLOC	110			010		011		Boot ROM interface location
12-13	RLEXT	00							Legacy mode
14-15	Reserved	00							—
16-27	Reserved	0000_0000_0000							—
28	TLE	0							Big-endian mode
29-31	Reserved	000							—

4.4 Clocking

The following external clock sources are utilized on the MPC8309:

- System clock (SYS_CLK_IN)
- Real-time clock (RTC_PIT_CLOCK)
- QE_CLK_IN

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

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

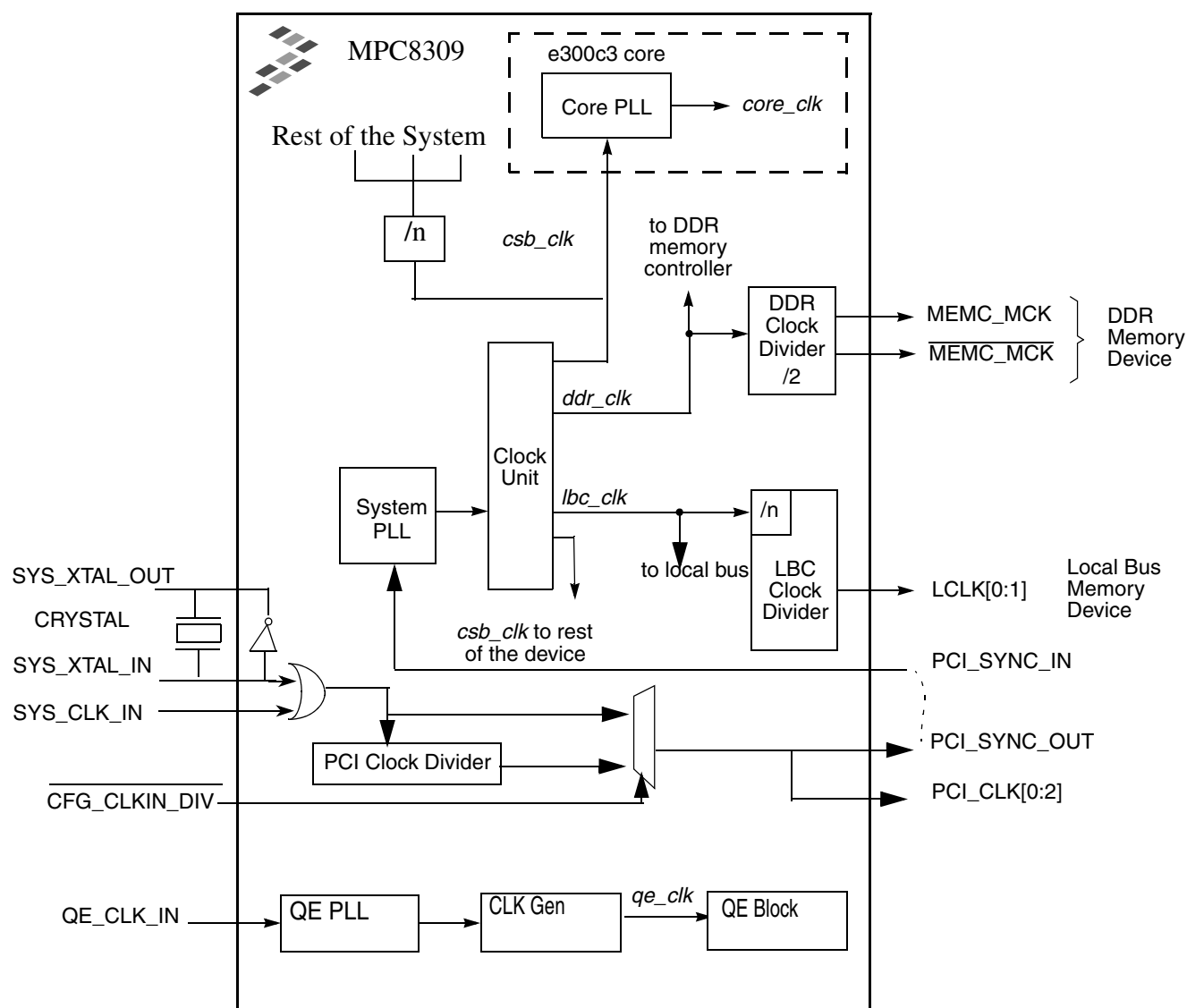


Figure 4-7. MPC8309 Clock Subsystem

4.4.1 System Clock Domains

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

- The coherent system bus clock (*csb_clk*)
- The internal clock for the DDR controller (*ddr_clk*)
- The internal clock for the local bus interface unit (*lbc_clk*)

The *csb_clk* frequency is derived as follows:

$$csb_clk = [SYS_CLK_IN] \times SPMF$$

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

The *qe_clk* frequency is determined by the QUICC Engine PLL multiplication factor (RCWL[CEPMF]) and the QUICC Engine PLL division factor (RCWL[CEPDF]) according to the following equations:

When QE_CLK_IN is the primary input clock, the *qe_clk* frequency is determined as follows:

$$qe_clk = (QE_CLK_IN \times CEPMF) \div (1 + CEPDF)$$

See [Section 4.3.2.1.3, “QUICC Engine PLL Multiplication Factor,”](#) and [Section 4.3.2.1, “Reset Configuration Word Low Register \(RCWLR\),”](#) for more information.

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

The local bus memory controller operates with a frequency equal to the frequency of *csb_clk*. Note that *lbc_clk* is not the external local bus frequency; *lbc_clk* passes through the LBC clock divider to create the external local bus clock output (LCLK). The LBC clock divider ratio is controlled by LCRR[CLKDIV]. See [Section 11.1.3.1, “eLBC Bus Clock and Clock Ratios,”](#) for more information.

In addition, some of the internal units may be required to be shut off or operate at lower frequency than the *csb_clk* frequency. These units have a default clock ratio that can be configured by a memory-mapped register after the device comes out of reset. [Table 4-22](#) specifies which units have a configurable clock frequency. For more information, see [Section 4.5.2.3, “System Clock Control Register \(SCCR\).”](#)

Table 4-22. Configurable Clock Units

Unit	Default Frequency	Options
I ² C	<i>csb_clk</i>	Off, <i>csb_clk</i> , <i>csb_clk</i> /2, <i>csb_clk</i> /3
USB DR	<i>csb_clk</i>	Off, <i>csb_clk</i> , <i>csb_clk</i> /2, <i>csb_clk</i> /3
DMA Engine 1	<i>csb_clk</i>	Off, <i>csb_clk</i> , <i>csb_clk</i> /2, <i>csb_clk</i> /3
DMA Engine 2	<i>csb_clk</i>	Off, <i>csb_clk</i>

Table 4-22. Configurable Clock Units (continued)

Unit	Default Frequency	Options
eSDHC	<i>csb_clk</i>	Off, <i>csb_clk</i> , <i>csb_clk/2</i> , <i>csb_clk/3</i>
QUICC Engine	<i>qe_clk</i>	Off, <i>qe_clk</i>

NOTE

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

4.5 Memory Map/Register Definitions

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

4.5.1 Reset Configuration Register Descriptions

The reset configuration and status registers are shown in [Table 4-23](#).

Table 4-23. Reset Configuration and Status Registers Memory Map

Address	Register	Access	Reset	Section/Page
Reset Configuration—Block Base Address 0x0_0900				
0x0_0900	Reset configuration word low register (RCWLR)	R	0xn _{nnn} _n _{nnn}	4.5.1.1/4-27
0x0_0904	Reset configuration word high register (RCWHR)	R	0xn _{nnn} _n _{nnn}	4.5.1.2/4-27
0x0_0908– 0x0_090C	Reserved, should be cleared	—	—	—
0x0_0910	Reset status register (RSR)	R/W	0x0000_000	4.5.1.3/4-28
0x0_0914	Reset mode register (RMR)	R/W	0x0000_0000	4.5.1.4/4-29
0x0_0918	Reset protection register (RPR)	R/W	0x0000_0000	4.5.1.5/4-30
0x0_091C	Reset control register (RCR)	R/W	0x0000_0000	4.5.1.6/4-30
0x0_0920	Reset control enable register (RCER)	R/W	0x0000_0000	4.5.1.7/4-31
0x0_0924	Reserved.	—	—	—
0x0_0928– 0x0_09FC	Reserved, should be cleared	—	—	—

4.5.1.1 Reset Configuration Word Low Register (RCWLR)

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

4.5.1.2 Reset Configuration Word High Register (RCWHR)

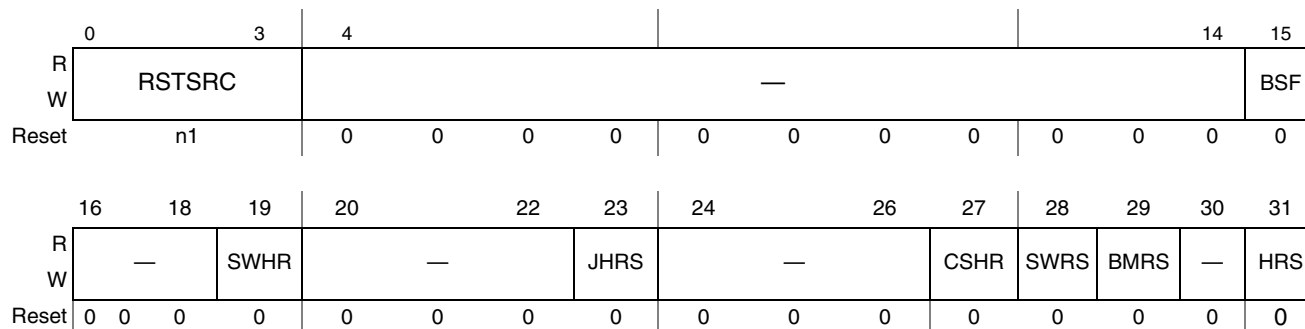
The reset configuration word high register (RCWHR) is shown in [Figure 4-4](#) and described in [Section 4.3.2.2, “Reset Configuration Word High Register \(RCWHR\).”](#)

4.5.1.3 Reset Status Register (RSR)

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

Address 0x0_0910

Access: Read/Write



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 4-8. Reset Status Register (RSR)

[Table 4-24](#) defines the reset status register bit fields.

Table 4-24. Reset Status Register Field Descriptions

Bits	Name	Description
0–3	RSTSRC	Reset configuration word source. Reflects the value of CFG_RESET_SOURCE input signal during the reset flow. See Section 4.3.1.1, “Reset Configuration Word Source.” Changing this field has no effect.
4–14	—	Reserved, should be cleared.
15	BSF	Boot sequencer fail. If set, indicates that the I ² C boot sequencer has failed while loading the reset configuration words. Cleared by writing a 1 to it (writing zero has no effect).
16–18	—	Reserved, should be cleared.
19	SWHR	Software hard reset. If set, indicates a software hard reset. SWHR is cleared by writing a 1 to it (writing zero has no effect).
20–22	—	Reserved
23	JHRS	JTAG HRESET. If set, indicates that a hard reset request arrived from the Test Module by a JTAG command. Cleared by writing a 1 to it (writing zero has no effect).
24–26	—	Reserved, should be cleared.
27	CSHR	Check stop reset status. When the core enters a checkstop state and the checkstop reset is enabled by the RMR[CSRE], CSRS is set and it remains set until software clears it. CSRS is cleared by writing a 1 to it (writing zero has no effect). 0 No enabled check stop reset event. 1 Enabled check stop reset event.
28	SWRS	Software watchdog reset status. When a software watchdog expire event (which causes a reset) is detected, SWRS is set and remains that way until the software clears it. SWRS is cleared by writing a 1 to it (writing zero has no effect). 0 No software watchdog reset event. 1 Software watchdog reset event.

Table 4-24. Reset Status Register Field Descriptions (continued)

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

4.5.1.4 Reset Mode Register (RMR)

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


Figure 4-9. Reset Mode Register (RMR)

[Table 4-25](#) describes the RMR fields.

Table 4-25. RMR Field Descriptions

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

4.5.1.5 Reset Protection Register (RPR)

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

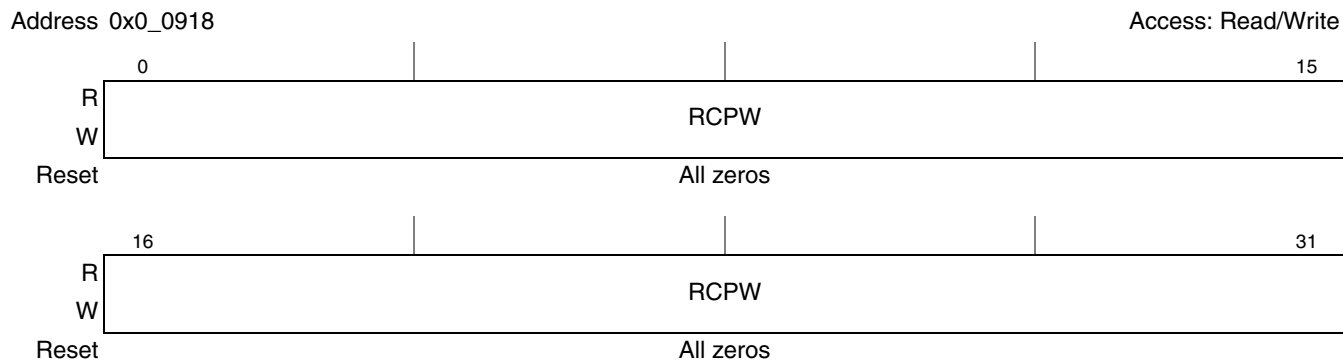


Figure 4-10. Reset Protection Register (RPR)

[Table 4-26](#) defines the bit fields of RPR.

Table 4-26. RPR Bit Descriptions

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

4.5.1.6 Reset Control Register (RCR)

RCR, shown in [Figure 4-11](#), can be used by software to initiate a hard reset sequence. To allow writing to this register, the user must enable it by writing the value 0x5253_5445 to the RPR.

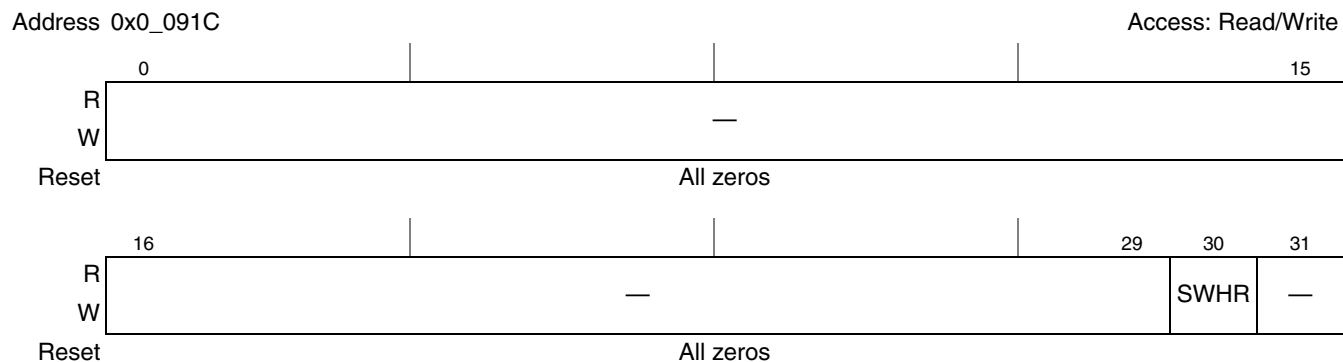


Figure 4-11. Reset Control Register (RCR)

Table 4-27 defines the bit fields of RCR.

Table 4-27. RCR Bit Settings

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

4.5.1.7 Reset Control Enable Register (RCER)

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

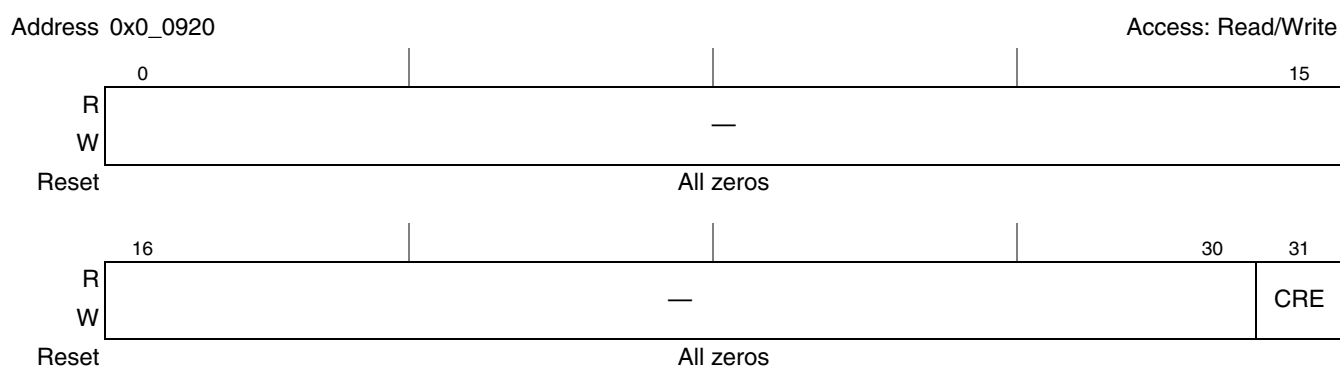


Figure 4-12. Reset Control Enable Register (RCER)

Table 4-28 defines the bit fields of RCER.

Table 4-28. 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 4-29.

Table 4-29. Clock Configuration Registers Memory Map

Address	Register	Access	Reset	Section/Page
Reset Configuration—Block Base Address 0x0_0A00				
0x0_0A00	System PLL mode register (SPMR)	R	0xn _{nnn} _n _{nnn}	4.5.2.1/4-32
0x0_0A04	Output clock control register (OCCR)	R/W	0x0000_C0C0	4.5.2.2/4-33

Table 4-29. Clock Configuration Registers Memory Map (continued)

Address	Register	Access	Reset	Section/Page
0x0_0A08	System clock control register (SCCR)	R/W	0x0542_0010	4.5.2.3/4-34
0x0_0A0C– 0x0_0AFC	Reserved, should be cleared	—	—	—

4.5.2.1 System PLL Mode Register (SPMR)

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

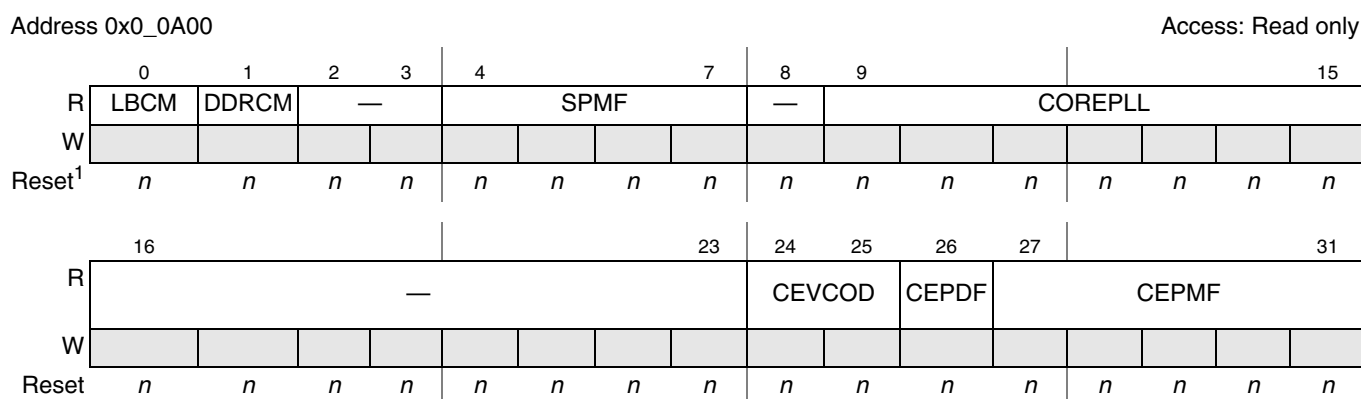


Figure 4-13. System PLL Mode Register

¹ See [Table 4-30](#) for reset values.

[Table 4-30](#) defines the system PLL mode register bit fields.

Table 4-30. System PLL Mode Register Bit Settings

Bits	Name	Meaning	Description
0	LBCM	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 4.3.2.1.2, “System PLL Configuration”
8	—	Reserved	—
9–15	COREPLL	Core PLL configuration.	For more information, see <i>MPC8309 PowerQUICC II Pro Integrated Communications Processor Hardware Specification</i> .
16–23	—	Reserved, should be cleared.	—

Table 4-30. System PLL Mode Register Bit Settings (continued)

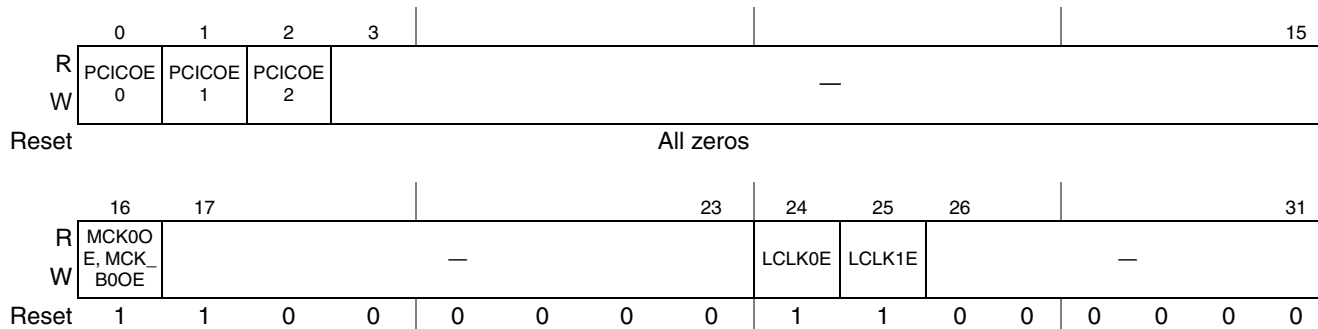
Bits	Name	Meaning	Description
24–25	CEVCOD	QUICC Engine PLL VCO division	Section 4.3.2.1, “Reset Configuration Word Low Register (RCWLR)”
26	CEPDF	QUICC Engine PLL division factor	Section 4.3.2.1, “Reset Configuration Word Low Register (RCWLR)”
27–31	CEPMF	QUICC Engine PLL multiplication factor	Section 4.3.2.1.3, “QUICC Engine PLL Multiplication Factor”

4.5.2.2 Output Clock Control Register (OCCR)

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

Address 0x0_0A04

Access: Read/Write


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

[Table 4-31](#) defines the bit fields of OCCR.

Table 4-31. OCCR Bit Settings

Bits	Name	Description
0	PCICOE0	PCI_CLK_OUT0 enable. 0 PCI_CLK_OUT0 signal is disabled (drive constant zero). 1 PCI_CLK_OUT0 signal is enabled to toggle.
1	PCICOE1	PCI_CLK_OUT1 enable. 0 PCI_CLK_OUT1 signal is disabled (drive constant zero). 1 PCI_CLK_OUT1 signal is enabled to toggle.
2	PCICOE2	PCI_CLK_OUT2 enable. 0 PCI_CLK_OUT2 signal is disabled (drive constant zero). 1 PCI_CLK_OUT2 signal is enabled to toggle.
3–15	—	Reserved, should be cleared
16	MCK0OE, MCK_B0OE	Enable/Disable MCK[0] pins clock out 0 Disable MCK[0] and MCK[0] 1 Enable MCK[0] and MCK[0]
17–23	—	Reserved, should be cleared

Table 4-31. OCCR Bit Settings (continued)

Bits	Name	Description
24	LCLK0E	Enable/Disable LCLK[0] pin clock out 0 Disable LCLK[0] 1 Enable LCLK[0]
25	LCLK1E	Enable/Disable LCLK[1] pin clock out 0 Disable LCLK[1] 1 Enable LCLK[1]
26–31	—	Reserved, should be cleared

4.5.2.3 System Clock Control Register (SCCR)

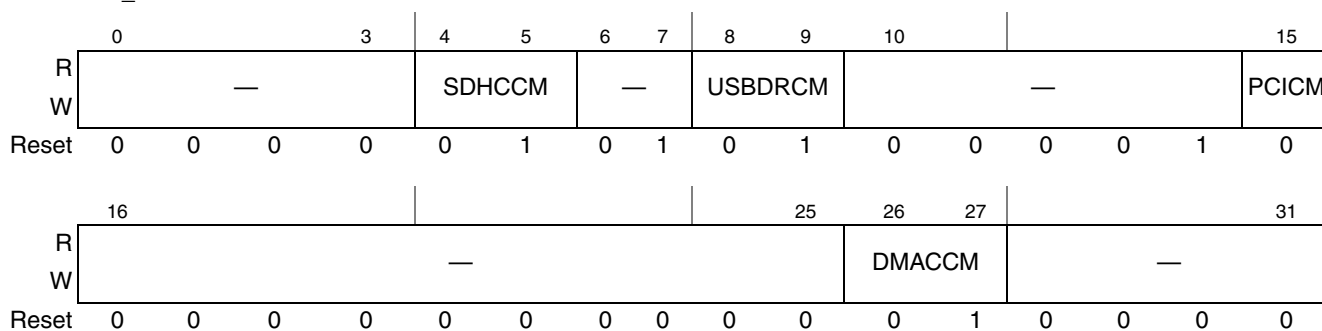
The system clock control register (SCCR), shown in [Figure 4-15](#), controls device units that have a configurable clock ratio.

NOTE

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

Address 0x0_0A08

Access: Read/Write


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

[Table 4-32](#) defines the bit fields of SCCR.

Table 4-32. SCCR Bit Descriptions

Bits	Name	Description
0–3	—	Reserved
4–5	SDHCCM	SDHC clock mode 00 SDHC core clock is disabled. 01 SDHC core clock/ <i>csb_clk</i> ratio is 1:1. 10 SDHC core clock/ <i>csb_clk</i> ratio is 1:2. 11 SDHC core clock/ <i>csb_clk</i> ratio is 1:3.
6–7	—	Reserved

Table 4-32. SCCR Bit Descriptions (continued)

Bits	Name	Description
8–9	USBDRCM	USB DR clock mode. It also controls I ² C1 clock ratio. 00 USB DR clock is disabled. I ² C1 clock is 1:1. 01 USB DR clock/ <i>csb_clk</i> ratio is 1:1. I ² C1 clock is 1:1. 10 USB DR clock/ <i>csb_clk</i> ratio is 1:2. I ² C1 clock is 1:2. 11 USB DR clock/ <i>csb_clk</i> ratio is 1:3. I ² C1 clock is 1:3.
10–14	—	Reserved
15	PCICM	PCI clock mode. Define the clock mode for all of the PCI complex - PCI and DMA. 0 PCI complex clocks are disabled. 1 PCI complex clocks are enabled.
16–25	—	Reserved
26–27	DMACCM	DMACCM. DMA Engine 1. 00 DMAC core clock is disabled. 01 DMAC core clock/ <i>csb_clk</i> ratio is 1:1. 10 DMAC core clock/ <i>csb_clk</i> ratio is 1:2. 11 DMAC core clock/ <i>csb_clk</i> ratio is 1:3.
28–31	—	Reserved



Chapter 5 System Boot

This chapter describes the overall initialization of the MPC8309.

5.1 Booting from On Chip ROM

Selecting on-chip ROM in boot ROM location causes the e300 CPU to fetch data from the on-chip ROM. The on-chip ROM is selected by configuring the POR config values of RCWHR appropriately as shown in [Table 5-1](#). For more information, see [Section 4.3.2.2.3, “Boot ROM Location](#).

The following different configurations are provided for boot from the on-chip ROM:

- Boot from eSDHC
- Boot from SPI

Table 5-1. Boot Source Selection for On-Chip ROM Boot- RCWHR Settings

Bits ¹	Name	Description
5	BMS	Boot Memory Space 1 : For booting from On-Chip ROM for both SPI and eSDHC boot.
9–11	ROMLOC	Boot ROM interface location 010 Boot from eSDHC 011 Boot from SPI
12–13	RLEXT	Boot ROM location extension. This bit combined with bit ROMLOC determines where the device boots from. 00 Legacy mode—allows for booting from on-chip peripherals. For booting from On Chip ROM, these bits must be set to 00.

¹ This table shows only relevant bits for On-Chip Boot configuration.

5.2 eSDHC Boot

This section explains how to boot from eSDHC. The following is discussed in this section:

- [Section 5.2.1, “Overview”](#)
- [Section 5.2.2, “Features”](#)
- [Section 5.2.3, “SD/MMC Card Data Structure”](#)
- [Section 5.2.4, “eSDHC Controller Initial Configuration”](#)
- [Section 5.2.5, “eSDHC Controller Boot Sequence”](#)
- [Section 5.2.6, “eSDHC Boot Error Handling”](#)

5.2.1 Overview

The MPC8309 is capable of loading initialization code from a memory device that is connected to the eSDHC controller interface. This device can be either a SD card, SDIO card, MMC card, or other variants compatible with these devices. The term SD/MMC is used when referring to the memory device.

Boot from eSDHC is supported by the MPC8309 using an on-chip ROM, which contains the basic eSDHC device driver and the code to perform block copy from SD/MMC to any target memory. Selecting on-chip ROM in boot ROM location (see [Table 5-1](#)) causes the e300 CPU to fetch data from the on-chip ROM. Prior to boot, the user should ensure that the SD/MMC card to boot from is inserted.

After the device has completed the reset sequence, if the ROM location selects the on-chip ROM eSDHC boot configuration, the e300 core starts to execute code from the internal on-chip ROM. The e300 core configures the eSDHC controller, enabling it to communicate with the external SD/MMC card. The SD/MMC card should contain a specific data structure with control words, device configuration information and initialization code. The on-chip ROM boot code uses the information from the SD/MMC card content to configure the device, and to copy the initialization code to a target memory device (for example, the DDR) through the eSDHC interface. After all the code has been copied, the e300 core starts to execute the code from the target memory device.

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

A more advanced option is for the on-chip ROM to only copy a small user-customized subroutine, which configures the eSDHC in an optimal way. The user-customised subroutine then copies the rest of the boot code potentially much faster than the on-chip ROM software can achieve. For example, the user-customised subroutine may utilize 4-bit or 8-bit eSDHC interfaces, or support new SD or MMC format revisions, or increase the external clock frequency based on knowledge of the exact frequency at which the MPC8309 is operating.

5.2.2 Features

The main features are as follows:

- Provides mechanism to load initialization code from the following external devices:
 - SD memory cards (up to and including version 2.0)
 - MMC, RS-MMC (up to and including version 4.0)
 - SDHC cards (SD High Capacity, from 4 GByte to 32 GByte)
- Boot from the following devices is not supported
 - SDIO and miniSDIO cards that are not SD combo cards and consequently have no memory
 - Locked (password-protected) SD/MMC cards
 - Secured Mode of SD cards (SD Card Specification Part 3: Security Specification)
- Simple data structure in SD/MMC card
- BOOT signature is checked to validate that the SD/MMC card contains valid code

- Supports variable code length in SD/MMC card
- Flexible target memory device
- Supports target memory configuration controlled by the user
- Only 1-bit operation is supported for boot (even if the SD/MMC card supports 4 or 8-bit parallel access).
- Initial setting uses a serial clock below 400 kHz; the SD/MMC internal registers are read by initialization code and parsed to determine the optimal clock frequency supported by the SD/MMC card inserted.
- High speed cards are supported (up to 50 MHz SD and 52 MHz MMC).
- Control word allows for user modification
- There must be precisely one device connected on the eSDHC bus. In particular, multiple MMC devices sharing the one bus are not supported.
- Compatible with FAT12/FAT16/FAT32 SD/MMC filesystems (provided ≤ 40 configuration words are used prior to copying user's code to system memory).
- Redundancy to support SD/MMC bad blocks, by searching for the BOOT signature in up to 24 blocks, and trying the next block if the BOOT signature is not found, or if a read CRC error is found.

5.2.3 SD/MMC Card Data Structure

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

Figure 5-1 shows the required SD/MMC card data structure.

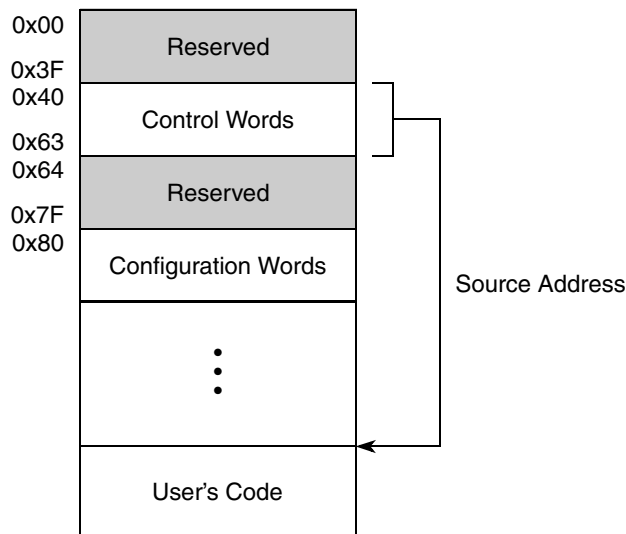


Figure 5-1. SD/MMC Card Data Structure

Table 5-2 describes the SD/MCC card data structure.

Table 5-2. SD/MMC Card Data Structure

Address	Data Bits [0–31]
0x00–0x3F	Reserved.
0x40–0x43	BOOT signature This location should contain the value 0x424F_4F54 , which is the ascii code for BOOT. The boot loader code will search for this signature. If the value in this location doesn't match the BOOT signature, it means that the SD/MMC card doesn't contain a valid user code. In such case the boot loader code will hang in an infinite loop.
0x44–0x47	Reserved
0x48–0x4B	User's code length Number of bytes in the user's code to be copied. This must be a multiple of the SD/MMC card's block size (and the user's code zero-padded if necessary to achieve that length). User's code length ≤ 2 GBytes.
0x4C–0x4F	Reserved
0x50–0x53	Source Address. Contains the starting address of the user's code as an offset from the SD/MMC card starting address. In Standard Capacity SD/MMC Cards, the 32-bit Source Address specifies the memory address in byte address format. This must be a multiple of the SD/MMC card's block size. In high capacity SD(SDHC) cards (> 2Gbytes), the 32-bit source address specifies the memory address in byte address format. However, it must be a multiple of block length, which is fixed to 512 bytes as per the SDHC specification.
0x54–0x57	Reserved
0x58–0x5B	Target Address Contains the target address in the system's local memory address space in which the user's code will be copied to.
0x5C–0x5F	Reserved
0x60–0x63	Execution Starting Address Contains the jump address in the system's local memory address space into the user's code first instruction to be executed.
0x64–0x67	Reserved
0x68–0x6B	N. Number of Config Address/Data pairs. Must be $1 \leq N \leq 1024$ (but is recommended to be as small as possible).
0x6C–0x7F	Reserved.
0x80–0x83	Config Address 1
0x84–0x87	Config Data 1
0x88–0x8B	Config Address 2
0x8C–0x8F	Config Data 2
...	
$0x80 + 8 \times (N-1)$	Config Address N ¹
$0x80 + 8 \times (N-1) + 4$	Config Data N (final Config Data N optional)

Table 5-2. SD/MMC Card Data Structure (continued)

Address	Data Bits [0–31]
	...
	User's code. Note that user's code must start on a 512-byte boundary.

¹ $N \leq 40$ if compatibility with FAT12/FAT16/FAT32 filesystems is required. Refer to [“Notes on Compatibility with FAT12/FAT16/FAT32 Filesystems”](#) for details.

5.2.3.1 Configuration Words Section

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

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

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

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

The Config Address structure is shown in [Figure 5-2](#).


Figure 5-2. Config Address Fields

[Table 5-3](#) defines the Config Address bits when CNT = 0 (address mode), and [Table 5-4](#) describes the Config Address bits when CNT = 1.

Table 5-3. Config Address Field Description, CNT = 0

Bits	Name	Description
0–29	Address	Address bits 0–29. The data in the Config Data field is copied by the e300 core to this address. The two least significant bits of the address (30:31) are always considered to be zero, as are the upper 4 bits of the 36-bit address.

Table 5-3. Config Address Field Description, CNT = 0 (continued)

Bits	Name	Description
30	—	Reserved. Must be zero.
31	CNT	Control. Select between Address mode and Control mode. 0 Address mode 1 Control mode

Table 5-4. Config Address Field Description, CNT = 1

Bits	Name	Description
0	EC	End Configuration. Indicates the end of the configuration stage. Valid only if bit CNT is set. 0 Not the last Config Address field. 1 The Last Config Address field. The e300 core will stop the configuration stage and start to copy the user's code. This must be set for Config Address Word N, and not be set for Config Address words prior to Config Address Word N.
1	DLY	Delay. Instruct the e300 core to perform delay according to the number that is specified in the adjacent Config Data field. The adjacent Config Data field provides the delay measured in terms of the number of 8 CSB clocks. Valid only if bit CNT is set. 0 No delay 1 Delay
2–30	—	Reserved. Must be zero.
31	CNT	Control. Select between Address mode and Control mode. 0 Address mode 1 Control mode Note: When CNT = 1, bits 0–29 select the control instruction. Only one bit in the range of bits 0–29 can be set at any specific control instruction. A control instruction with bits 0–29 all cleared is also illegal.

5.2.3.2 Notes on Compatibility with FAT12/FAT16/FAT32 Filesystems

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

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

Note that FAT12, FAT16 and FAT32 standards impose additional requirements on the data structures that must be present on the SD/MMC card, such as Partition Tables and a fixed Signature Word at the end of

the Master Boot Record. These features are not interpreted or required by the eSDHC boot process, and are outside the scope of this document.

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

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

5.2.4 eSDHC Controller Initial Configuration

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

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

- Address Invariant Mode (eSDHC.PROTCTL[EMODE] = 10)
- SDHC_DAT[3] does not monitor card insertion.
- 1-bit Mode (eSDHC.PROTCTL[DTW] = 00)
- SDCLK at 400 kHz or below, but higher than 100 kHz .
- There must be precisely one device connected on the eSDHC bus (and this device must be inserted prior to boot). Multiple MMC devices sharing the one bus are not supported.
- The eSDHC DMA engine is not used for Control or Configuration Word accesses; instead, all eSDHC data transfers are initiated by the processor core polling eSDHC.PRSTAT[BRR] and accessing data through the DATPORT register (XFERTYP[DMAEN] = 0). The eSDHC DMA engine is used for user code accesses.

5.2.5 eSDHC Controller Boot Sequence

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

1. The eSDHC controller is configured as per [Section 5.2.4, “eSDHC Controller Initial Configuration.”](#)
2. Card-detect.
3. The SD/MMC card is reset.
4. SD/MMC card voltage validation is performed.
5. SD/MMC card identification.
6. With CMD9, the CSD (Card-Specific Data) register of the SD/MMC card is read.

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

5.2.6 eSDHC Boot Error Handling

If at any stage the boot loader code detects an error and cannot continue, it will hang. This may occur in any of the following scenarios:

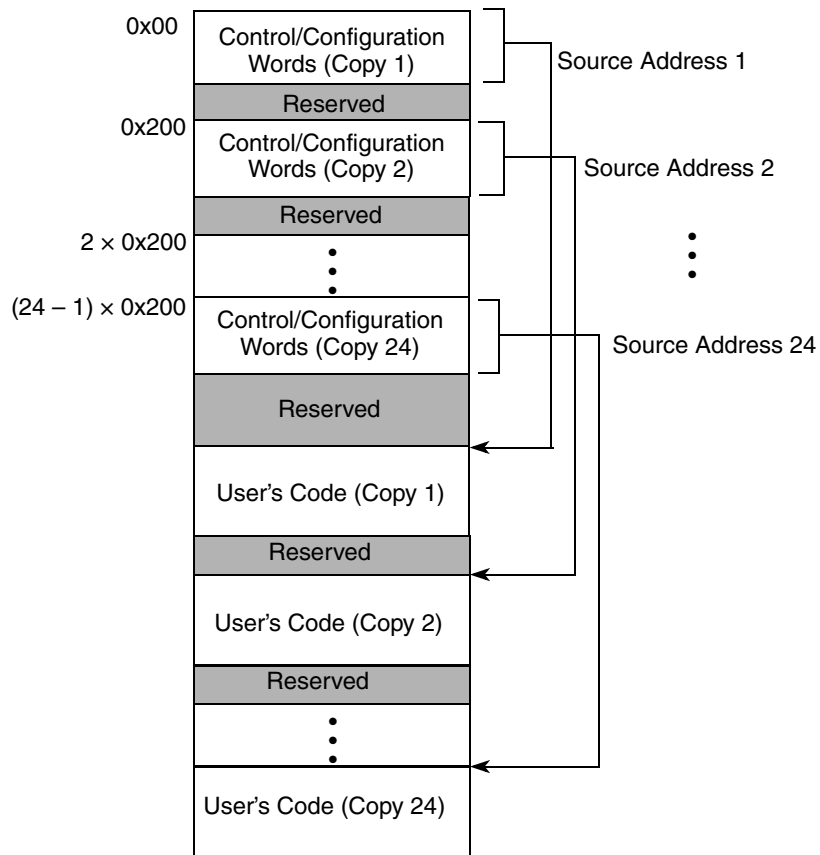
- BOOT signature not found at offset 0x40 or CRC error on any of the data read by the eSDHC 24 times.
- Timeout while waiting for the SD/MMC card to respond at any stage.
- No card inserted.
- Incorrect type of card inserted that is not supported for boot (such as CE-ATA).
- There is no common protocol,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

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

For reference, [Figure 5-3](#) shows an example SD/MMC memory card data structure that can be used for maximum SD/MMC card data redundancy.

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

Each copy of the control/configuration words would generally be identical except for the pointer to the source address (offset 0x50) of the SD/MMC card, which may be different for each copy. If the user's code section is sufficiently large that 24 copies of it do not fit in the capacity of the SD/MMC card (or if the SD/MMC card capacity must also be utilized for functional features other than system boot), then it may still be desirable to still support up to 24 copies of the control/configuration words, but only have them reference a limited number of user's code sections.



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

Figure 5-3. SD/MMC Card Data Structure for Maximum Redundancy

5.3 SPI Boot ROM

This section explains how to boot from SPI. The following is discussed in this section:

- [Section 5.3.1, “Overview”](#)
- [Section 5.3.2, “Features”](#)
- [Section 5.3.3, “EEPROM Data Structure”](#)
- [Section 5.3.4, “SPI Controller Configuration”](#)

5.3.1 Overview

The MPC8309 is capable of loading initialization code from a memory device that is connected to the SPI controller interface. This device can be either an EEPROM or a serial flash with an SPI-compatible interface. The term EEPROM will be used when referring to the memory device.

Boot from SPI is supported by the MPC8309 using an on-chip ROM which contains the basic SPI device driver and the code to perform block copy from SPI EPROM to any target memory. Selecting on-chip ROM in boot ROM location, see [Table 5-1](#), causes the e300 CPU to fetch data from the on-chip ROM. After the device has completed the reset sequence, if the ROM location selects the on-chip ROM SPI Boot configuration, the e300 core starts to execute code from the internal on-chip ROM. The e300 core configures the SPI controller, enabling it to communicate with the external EEPROM. The EEPROM should contain a specific data structure with control words, device configuration information and initialization code. The on-chip ROM boot code uses the information from the EEPROM content to configure the device, and to copy the initialization code to a target memory device (for example, the DDR) through the SPI interface. After all the code has been copied, the e300 core starts to execute the code from the target memory device.

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

A more advanced option is for the on-chip ROM to only copy a small user-customised subroutine, which configures the SPI in an optimal way. The user-customised subroutine then copies the rest of the boot code potentially much faster than the on-chip ROM software can achieve.

5.3.2 Features

The main features are as follows:

- Provides mechanism to load initialization code from external SPI EEPROM
- Simple data structure in SPI EEPROM
- BOOT signature will be checked to validate that the EEPROM contains valid code
- Supports variable code length in EEPROM
- Flexible target memory device
- Supports target memory configuration controlled by the user

5.3.3 EEPROM Data Structure

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

Figure 5-4 shows the required SPI EEPROM data structure.

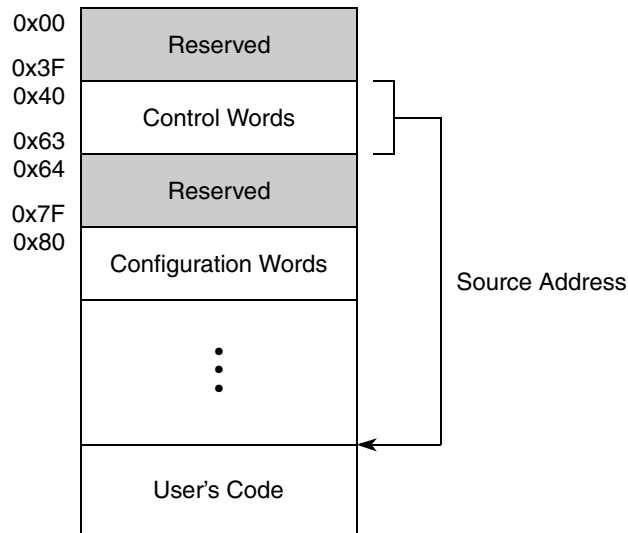


Figure 5-4. SPI EEPROM Data Structure

Table 5-5 describes the SPI EEPROM data structure.

Table 5-5. SPI EEPROM Data Structure

Address	Data Bits [0–31]
0x00–0x3F	Reserved.
0x40–0x43	BOOT signature. This location should contain the value 0x424F_4F54, which is the ASCII code for BOOT. The SPI loader code will search for this signature, initially in 24-bit addressable mode. If the value in this location doesn't match the BOOT signature, then the EEPROM is accessed again, but in 16-bit mode. If the value in this location still does not match the BOOT signature, it means that the SPI device doesn't contain a valid user code. In such case the SPI loader code will hang.
0x44–0x47	Reserved
0x48–0x4B	User's code length. Number of bytes in the user's code to be copied. Must be a multiple of 4. $4 \leq \text{User's code length} \leq 2 \text{ Gbytes}$.
0x4C–0x4F	Reserved
0x50–0x53	Source Address. Contains the starting address of the user's code as an offset from the EEPROM starting address.
0x54–0x57	Reserved
0x58–0x5B	Target Address. Contains the target address in the system's local memory address space in which the user's code will be copied to.
0x5C–0x5F	Reserved
0x60–0x63	Execution Starting Address. Contains the jump address in the system's local memory address space into the user's code first instruction to be executed.
0x64–0x67	Reserved

Table 5-5. SPI EEPROM Data Structure (continued)

Address	Data Bits [0–31]
0x68–0x6B	N. Number of Config Address/Data pairs. Must be ≤ 1024 (but is recommended to be as small as possible).
0x6C–0x7F	Reserved.
0x80–0x83	Config Address 1
0x84–0x87	Config Data 1
0x88–0x8B	Config Address 2
0x8C–0x8F	Config Data 2
...	
0x80 + 8×(N-1)	Config Address N
0x80 + 8×(N-1) + 4	Config Data N (Final Config Data N optional)
...	
	User's Code

5.3.3.1 Configuration Words Section

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

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

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

The Config Address structure is shown in [Figure 5-5](#).



Figure 5-5. Config Address Fields

Table 5-6 shows the Config Address field description, CNT = 0.

Table 5-6. Config Address Field Description, CNT = 0

Bits	Name	Description
0–29	Address	Address bits 0–29. The data in the Config Data field is copied by the e300 core to this address.
30	—	Reserved. Must be zero.
31	CNT	Control. Select between Address mode and Control mode. 0 Address mode 1 Control mode

Table 5-7 shows the Config Address field description, CNT = 1.

Table 5-7. Config Address Field Description, CNT = 1

Bits	Name	Description
0	EC	End Configuration. Indicates the end of the configuration stage. Valid only if bit CNT is set. 0 Not the last Config Address field. 1 The Last Config Address field. The e300 core will stop the configuration stage and start to copy the user's code. This must be set for Config Address Word N, and not be set for Config Address words prior to Config Address Word N.
1	DLY	Delay. Instruct the e300 core to perform delay according to the number that is specified in the adjacent Config Data field. The adjacent Config Data field provides the delay measured in terms of the number of 8 CSB clocks. Valid only if bit CNT is set. 0 No delay. 1 Delay.
2	CF	Change frequency. Instruct the e300 core to perform sequence of operations to setup the SPI CS1 mode register with the frequency related (PM and DIV16) bits as defined by the user. The adjacent Config Data field will be written to the SPI mode register. Software will use DIV16 and PM bits and mask all other bits such that they will not change. Software will perform the necessary steps which are required by the SPI controller before and after changing the SPI mode register. This only takes effect after all of the Configuration and Control words have been read.
3–30	—	Reserved. Must be zero.
31	CNT	Control. Select between Address mode and Control mode. 0 Address mode 1 Control mode Note: When CNT = 1, bits 0-29 select the control instruction. Only one bit in the range of bits 0-29 can be set at any specific control instruction. A control instruction with bits 0-29 all cleared is also illegal.

5.3.4 SPI Controller Configuration

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

The SPI controller is configured to operate in master mode. The $\overline{\text{SPISEL}}$ input should be forced inactive by an external pull up. The SPI chip select ($\overline{\text{SPISEL_BOOT}}$) must be connected to the EEPROM $\overline{\text{CS}}$ and selectively enables the EEPROM.

Figure 5-6 shows the external signal connection.

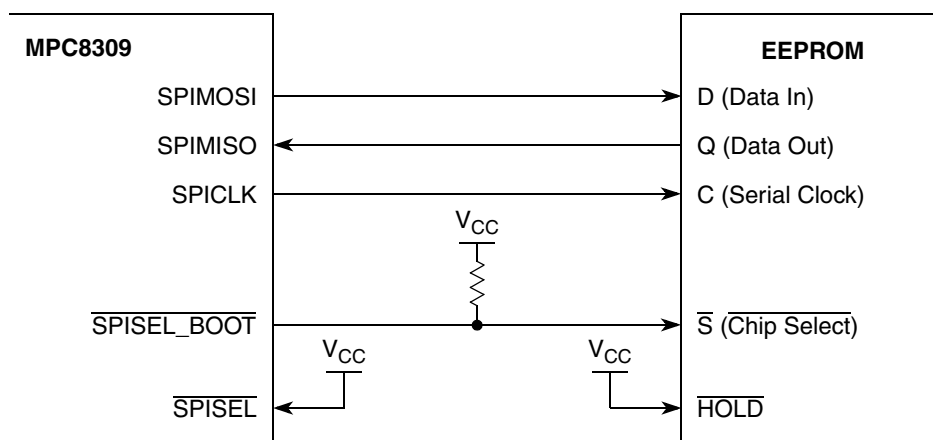


Figure 5-6. External Signal Connection

Chapter 6

System Configuration

6.1 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 6.2, “Local Memory Map Overview and Example”](#)
- [Section 6.3, “System Configuration”](#)
- [Section 6.4, “Software Watchdog Timer \(WDT\)”](#)
- [Section 6.5, “Real Time Clock Module \(RTC\)”](#)
- [Section 6.6, “Periodic Interval Timer \(PIT\)”](#)
- [Section 6.7, “General-Purpose Timers \(GTM\)”](#)
- [Section 6.8, “Power Management Control \(PMC\)”](#)

6.2 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 seven local access windows. Each of these windows maps a region of memory to a particular target interface, such as the DDR SDRAM controller. Note that the local access windows do not perform any address translation. The size of each window can be configured from 4 Kbytes to 2 Gbytes. Each local access window is assigned to a specific target interface as specified in [Table 6-1](#).

Table 6-1. Local Access Windows Target Interface

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

Table 6-1. Local Access Windows Target Interface (continued)

Window Number	Target Interface	Comments
7	DDR2 SDRAM	—
8	DDR2 SDRAM	—

Figure 6-1 shows an example memory map.

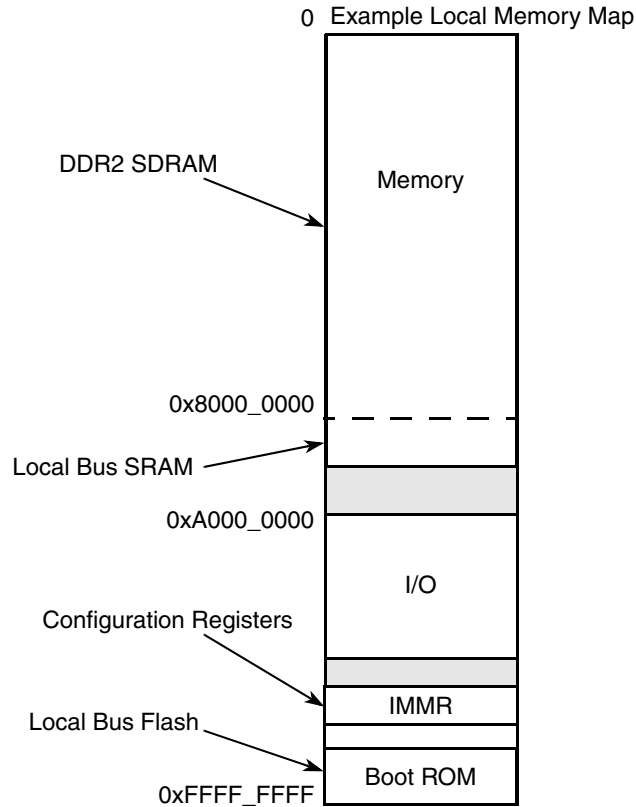


Figure 6-1. Local Memory Map Example

Table 6-2 shows one example of local access window settings.

Table 6-2. Local Access Windows Example

Window	Base Address	Size	Target Interface
5	0x0000_0000	2 Gbytes	DDR2 SDRAM
2	0x8000_0000	1 Mbyte	Local bus
7	0xA000_0000	1 Mbyte	PCI
3	0xC000_0000	256 Mbytes	Local bus
0	0xFF40_0000	2 Mbyte	Configuration registers (IMMR)

Table 6-2. Local Access Windows Example (continued)

Window	Base Address	Size	Target Interface
1	0xFF80_0000	8 Mbytes	Local bus boot ROM Flash
6,4	Unused		

In this example, the local access window of the boot ROM is defined as window number 1, on a local bus device, in the highest 8 Mbytes of memory as set by the reset configuration word high during the reset sequence (see [Section 4.3.2.2.3, “Boot ROM Location”](#)) and [Section 6.2.4.3.1, “LBLAWBAR0\[BASE_ADDR\] Reset Value.”](#) The local access window, which describes the range of memory used for memory-mapped registers (IMMR), is a fixed 2-Mbyte space pointed to by the IMMRBAR register, using its default value (0xFF40_0000). See [Section 6.2.4.1, “Internal Memory Map Registers Base Address Register \(IMMRBAR\).”](#)

6.2.1 Address Translation and Mapping

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

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

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

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

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

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

Inbound windows perform address translation from the external address spaces to the local address space.

All of the configuration registers that define mapping of local access windows follow the same register format. [Table 6-3](#) summarizes the general format of these window definitions.

Table 6-3. 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 2-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.

6.2.2 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 2 Mbytes, 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 6.2.4.1, "Internal Memory Map Registers Base Address Register \(IMMRBAR\)."](#)

NOTE

Although it is legal to use the 2-Mbyte space consecutive to the 2 Mbytes of the IMMR (for example, if IMMRBAR is 0xFF40_0000, the 2-Mbyte address space consecutive to it is 0xFF60_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.

6.2.3 Local Access Windows

As demonstrated in the address map overview in [Section 6.2, "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. PCIThis includes addresses that are mapped by PCI Express inbound windows.

The local access window registers exist as part of the local access block in the system configuration registers. See [Section 6.3.2, "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.

6.2.3.1 Local Access Register Memory Map

[Table 6-4](#) shows the memory map for the local access registers.

Table 6-4. Local Access Register Memory Map

Local Access—Block Base Address 0x0_0000				
Local Memory Offset (Hex)	Register	Access	Reset	Section/Page
0x0_0000	Internal memory map base address register (IMMRBAR)	R/W	0xFF40_0000	6.2.4.1/6-6
0x0_0004	Reserved	—	—	—
0x0_0008	Alternate configuration base address register (ALTCBAR)	R/W	0x0000_0000	6.2.4.2/6-7
0x0_000C–0x0_001C	Reserved	—	—	—
0x0_0020	eLBC local access window 0 base address register (LBLAWBAR0)	R/W	0x0000_0000 ¹	6.2.4.3/6-8
0x0_0024	eLBC local access window 0 attribute register (LBLAWAR0)	R/W	0x0000_0000 ²	6.2.4.4/6-9
0x0_0028	eLBC local access window 1 base address register (LBLAWBAR1)	R/W	0x0000_0000	6.2.4.3/6-8
0x0_002C	eLBC local access window 1 attribute register (LBLAWAR1)	R/W	0x0000_0000	6.2.4.4/6-9
0x0_0030	eLBC local access window 2 base address register (LBLAWBAR2)	R/W	0x0000_0000	6.2.4.3/6-8
0x0_0034	eLBC local access window 2 attribute register (LBLAWAR2)	R/W	0x0000_0000	6.2.4.4/6-9
0x0_0038	eLBC local access window 3 base address register (LBLAWBAR3)	R/W	0x0000_0000	6.2.4.3/6-8
0x0_003C	eLBC local access window 3 attribute register (LBLAWAR3)	R/W	0x0000_0000	6.2.4.4/6-9
0x0_0040–0x0_005C	Reserved	—	0x0000_0000	—
0x0_0060	PCI local access window 0 base address register (PCILAWBAR0)	R/W	0x0000_0000 ³	6.2.4.5/6-10
0x0_0064	PCI local access window 0 attribute register (PCILAWAR0)	R/W	0x0000_0000 ⁴	6.2.4.6/6-11
0x0_0068	PCI local access window 1 base address register (PCILAWBAR1)	R/W	0x0000_0000 ⁵	6.2.4.5/6-10
0x0_006C	PCI local access window 1 attribute register (PCILAWAR1)	R/W	0x0000_0000	6.2.4.6/6-11
0x0_0070–0x0_00	Reserved	—	—	—
0x0_00A0	DDR2 local access window 0 base address register (DDRLAWBAR0)	R/W	0x0000_0000 ⁶	6.2.4.7/6-12
0x0_00A4	DDR2 local access window 0 attribute register (DDRLAWAR0)	R/W	0x0000_0000 ⁷	6.2.4.8/6-13
0x0_00A8	DDR2 local access window 1 base address register (DDRLAWBAR1)	R/W	0x0000_0000	6.2.4.7/6-12
0x0_00AC	DDR2 local access window 1 attribute register (DDRLAWAR1)	R/W	0x0000_0000	6.2.4.8/6-13
0x0_00B0–0x0_00FC	Reserved	—	—	—

¹ Depends on reset configuration word high values. See [Section 6.2.4.3.1, “LBLAWBAR0\[BASE_ADDR\] Reset Value,”](#) for details.

² Depends on reset configuration word high values. See [Section 6.2.4.4.1, “LBLAWAR0\[EN\] and LBLAWAR0\[SIZE\] Reset Value,”](#) for details.

³ Depends on reset configuration word high values.

System Configuration

- ⁴ Depends on reset configuration word high values.
- ⁵ Depends on reset configuration word high values. See [Section 6.2.4.6.1, “PCILAWAR0\[EN\] and PCILAWAR0\[SIZE\] Reset Value,”](#) for details.
- ⁶ Depends on reset configuration word high values. See [Section 6.2.4.7.1, “DDRLAWBAR0\[BASE_ADDR\] Reset Value,”](#) for details.
- ⁷ Depends on reset configuration word high values. See [Section 6.2.4.8.1, “DDRLAWAR0\[EN\] and DDRLAWAR0\[SIZE\] Reset Value,”](#) for details.

6.2.4 Local Access Register Descriptions

This section describes the local access registers.

6.2.4.1 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 2-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.

6.2.4.1.1 Updating IMMRBAR

Updates to IMMRBAR that relocate the entire 2-Mbyte region of the internal memory block require special treatment. The effect of the update must be guaranteed to be visible by the mapping logic before an access to the new location is seen. To make sure this happens, the following guidelines should be followed:

- IMMRBAR should be updated during initial configuration of the device when only one host or controller has access to the device as follows:
 - If the core is initializing the device, it should set IMMRBAR to the desired final location before enabling other I/O devices to access the device.
- When the e300 core is writing to IMMRBAR, it should use the following sequence:
 - Read the current value of IMMRBAR using a load word instruction followed by an **isync**. This forces all accesses to configuration space to complete.
 - Write the new value to IMMRBAR.
 - Perform a load of an address that does not access configuration space or the on-chip SRAM, but has an address mapping already in effect (for example, boot ROM). Follow this load with an **isync**.
 - Read the contents of IMMRBAR from its new location, followed by another **isync**.

The IMMRBAR is shown in [Figure 6-2](#).

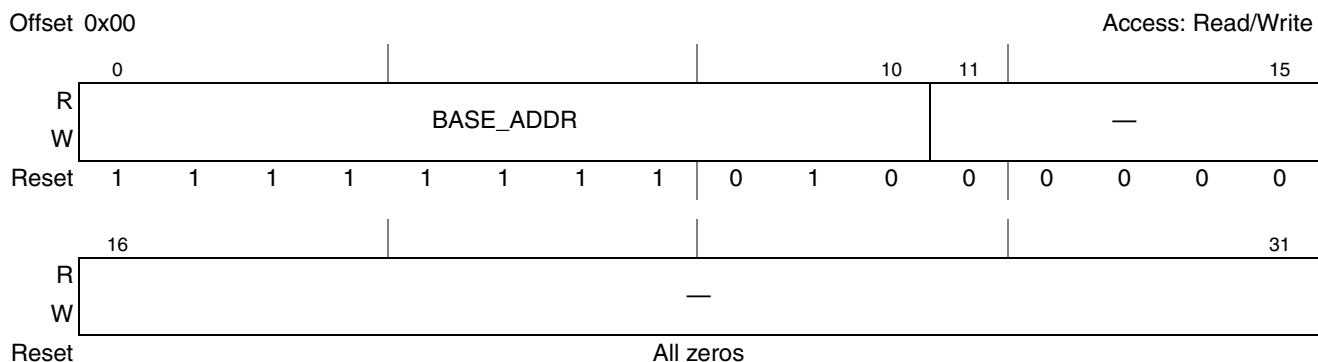


Figure 6-2. Internal Memory Map Registers' Base Address Register (IMMRBAR)

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

Table 6-5. IMMRBAR Bit Settings

Bits	Name	Description
0–10	BASE_ADDR	Identifies the 11 most-significant address bits of the base of the 2-Mbyte internal memory window.
11–31	—	Reserved. Software must write all zeros.

6.2.4.2 Alternate Configuration Base Address Register (ALTCBAR)

The alternate configuration base address register (ALTCBAR) is used to define the base address for an alternate 2-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 21 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 2-Mbyte block at a time. See [Section 17.4.5, “Boot Sequencer Mode,”](#) for more information.

NOTE

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

The alternate configuration base address register is shown in [Figure 6-3](#).

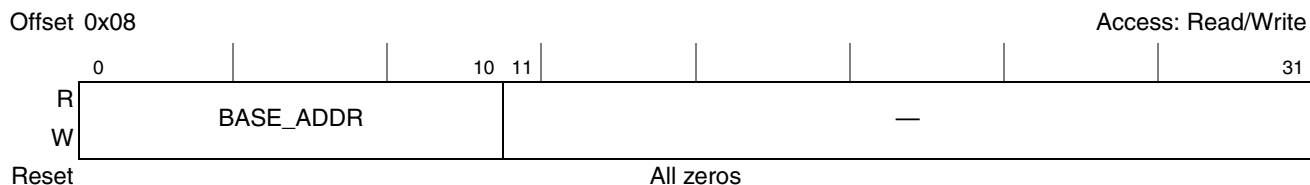


Figure 6-3. Alternate Configuration Base Address Register (ALTCBAR)

Table 6-6 defines the bit fields of ALTCBAR.

Table 6-6. ALTCBAR Bit Settings

Bits	Name	Description
0–10	BASE_ADDR	Identifies the 11 most-significant address bits of an alternate base address used for boot sequencer configuration accesses.
11–31	—	Reserved. Write has no effect, read returns 0.

6.2.4.3 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 6-4.



Figure 6-4. LBC Local Access Window *n* Base Address Registers (LBLAWBAR0–LBLAWBAR3)

¹ The LBLAWBAR0[BASE_ADDR] reset value depends on the reset configuration word high values. See Section 6.2.4.3.1, “LBLAWBAR0[BASE_ADDR] Reset Value,” for a detailed description.

Table 6-7 defines the bit fields of LBLAWBAR0–LBLAWBAR3.

Table 6-7. LBLAWBAR0–LBLAWBAR3 Bit Settings

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

6.2.4.3.1 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.

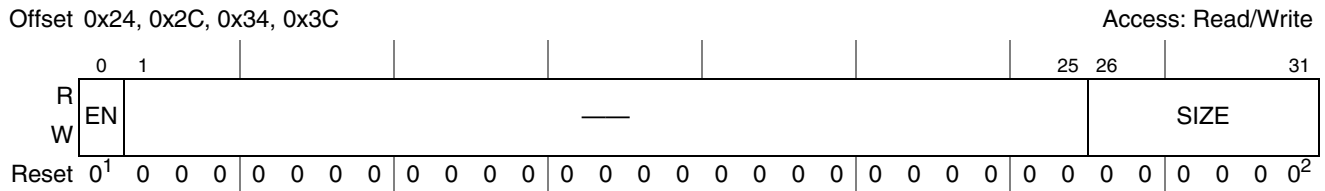
Table 6-8 defines the reset value of LBLAWBAR0[BASE_ADDR].

Table 6-8. LBLAWBAR0[BASE_ADDR] Reset Value

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

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

The LBC local access window *n* attributes registers (LBLAWAR0–LBLAWAR3) are shown in Figure 6-5.



- 1 The LBLAWAR0[EN] reset value depends on the reset configuration word high values. See Section 6.2.4.4.1, “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 6.2.4.4.1, “LBLAWAR0[EN] and LBLAWAR0[SIZE] Reset Value,” for a detailed description.

Figure 6-5. LBC Local Access Window *n* Attributes Registers (LBLAWAR0–LBLAWAR3)

Table 6-9 defines the bit fields of LBLAWAR0–LBLAWAR3.

Table 6-9. LBLAWAR0–LBLAWAR3 Bit Settings

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

6.2.4.4.1 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 6-10 defines the reset value for LBLAWAR0[EN].

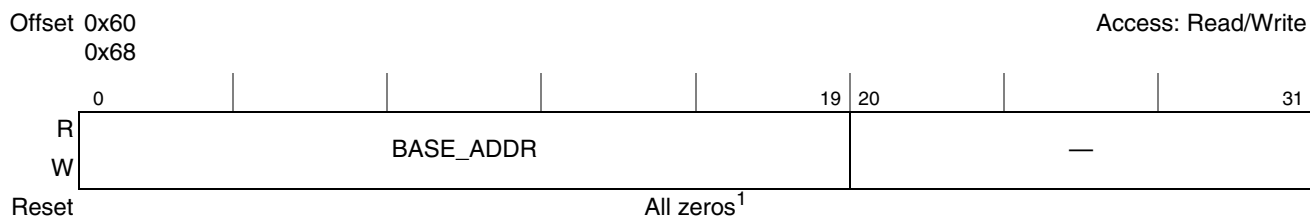
Table 6-10. LBLAWAR0[EN] Reset Value

RCWHR[ROMLOC]	RLEXT ¹	LBLAWAR0[EN] Reset Value	Description
001	01	1	e300 core boot performed from a local bus device. Local bus 8-Mbyte ($2^{(22+1)}$) local access window is enabled.
101	00		
	01		
110	00		
others		0	e300 core boot not performed from a local bus device.

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

6.2.4.5 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 6-6.



¹ The reset value of PCILAWBAR0[BASE_ADDR] depends on the reset configuration word high values. See Section 6.2.4.5.1, “PCILAWBAR0[BASE_ADDR] Reset Value,” for a detailed description.

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

Table 6-11 defines the bit fields of PCILAWBAR0–PCILAWBAR1.

Table 6-11. 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 <i>n</i> . The specified base address should be aligned to the window size, as defined by PCILAWAR <i>n</i> [SIZE].
20–31	—	Reserved. Write has no effect, read returns 0.

6.2.4.5.1 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.

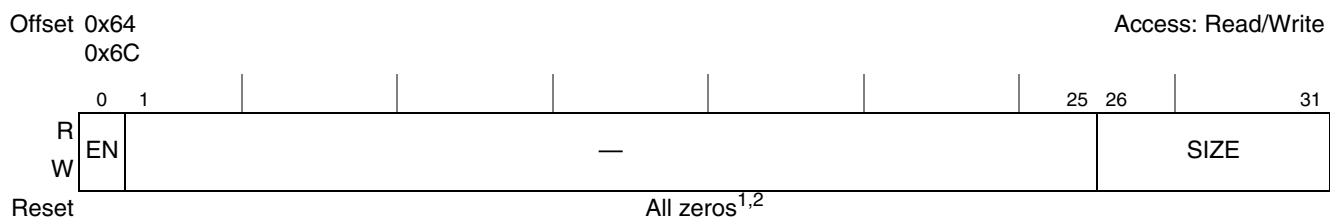
Table 6-12 defines the reset value of PCILAWBAR0[BASE_ADDR].

Table 6-12. PCILAWBAR0[BASE_ADDR] Reset Value

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

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

The PCI local access window n attributes registers (PCILAWAR0–PCILAWAR1) are shown in Figure 6-7.



¹ The reset value of PCILAWAR0[EN] depends on the reset configuration word high values. See Section 6.2.4.6.1, “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 6.2.4.6.1, “PCILAWAR0[EN] and PCILAWAR0[SIZE] Reset Value,” for a detailed description

Figure 6-7. PCI Local Access Window n Attributes Registers (PCILAWAR0–PCILAWAR1)

Table 6-13 defines the bit fields of PCILAWAR0–PCILAWAR1.

Table 6-13. 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 $2^{(\text{SIZE}+1)}$ bytes. 000000–001010 Reserved. Window is undefined. 001011 4 Kbytes 001100 8 Kbytes 001101 16 Kbytes $2^{(\text{SIZE}+1)}$ bytes 011110 2 Gbytes 011111–111111 Reserved. Window is undefined.

6.2.4.6.1 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.

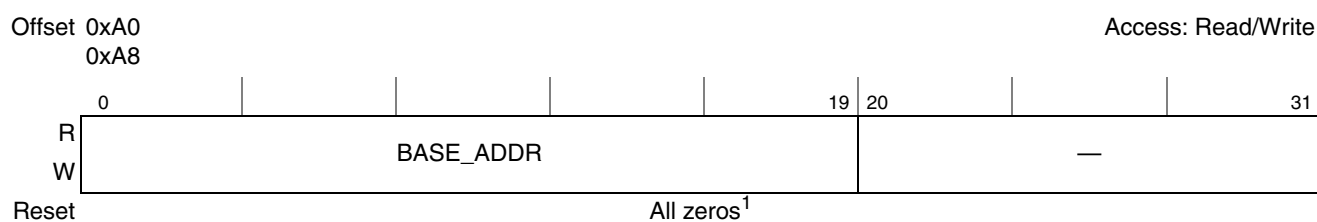
Table 6-14 defines the reset value of PCILAWAR0[EN].

Table 6-14. PCILAWAR0[EN] Reset Value

RCWHR[ROMLOC]	PCILAWR0[EN] Reset Value	Description
000	0	e300 core boot not performed from a PCI device.

6.2.4.7 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 6-8.



¹ The reset value of DDRLAWBAR0[BASE_ADDR] depends on the reset configuration word high values. See Section 6.2.4.7.1, “DDRLAWBAR0[BASE_ADDR] Reset Value,” for a detailed description

Figure 6-8. DDR Local Access Window *n* Base Address Registers (DDRLAWBAR0–DDRLAWBAR1)

Table 6-15 defines the bit fields of DDRLAWBAR0–DDRLAWBAR1.

Table 6-15. 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 <i>n</i> . The specified base address should be aligned to the window size, as defined by DDRLAWAR <i>n</i> [SIZE].
20–31	—	Reserved. Write has no effect, read returns 0.

6.2.4.7.1 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.

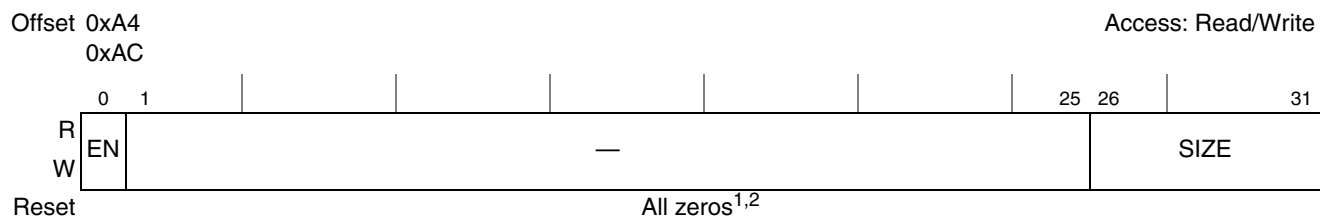
Table 6-16 defines the reset value DDRLAWBAR0.

Table 6-16. DDRLAWBAR0[BASE_ADDR] Reset Value

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

6.2.4.8 DDR Local Access Window n Attributes Registers (DDRLAWAR0–DDRLAWAR1)

The DDR local access window n attributes registers (DDRLAWAR0–DDRLAWAR1) are shown in Figure 6-9.



¹ The reset value of DDRLAWAR0[EN] depends on the reset configuration word high values. See Section 6.2.4.8.1, “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 6.2.4.8.1, “DDRLAWAR0[EN] and DDRLAWAR0[SIZE] Reset Value,” for a detailed description.

Figure 6-9. DDR Local Access Window n Attributes Registers (DDRLAWAR0–DDRLAWAR1)

Table 6-17 defines the bit fields of DDRLAWAR0–DDRLAWAR1.

Table 6-17. 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.

6.2.4.8.1 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 6-18 defines the reset value DDRLAWAR0[EN] and DDRLAWAR0[SIZE].

Table 6-18. DDRLAWAR0[EN] Reset Value

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

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

6.2.5 Precedence of Local Access Windows

If two local access windows overlap, the lower numbered window takes precedence (see Table 6-19 for window numbers). For instance, if two windows are set up as shown in Table 6-19, local access window 1 governs the mapping of the 1-Mbyte region from 0x7FF0_0000 to 0x7FFF_FFFF, even though the window described in local access window 5 also encompasses that memory region.

Table 6-19. Overlapping Local Access Windows

Window	Base Address	Size	Target Interface
1	0x7FF0_0000	1 Mbyte	Local bus
5	0x0000_0000	2 Gbytes	DDR SDRAM

6.2.6 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 e300 core, the read of LBLAWAR3 should be followed by an **isync** instruction.

6.2.7 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 PCI Express interface has outbound address translation units that map the local address into an external address space.

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

6.2.8 hasas **Outbound Address Translation and Mapping Windows**

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

The PCI controller has six outbound windows plus a default window. See [Section 4.5, “Memory Map/Register Definitions,”](#) for a detailed description of the PCI outbound windows.

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

6.2.9.1 **PCI Express Inbound Windows**

The PCI Express controller has four inbound windows. In the PCI Express EP (End Point) mode, these windows are same as the base address registers in the PCI Express programming model. In the PCI Express RC (Root Complex) mode, there are four separate registers, PEX_RCIWBARL[0:3].

6.2.9.2 **PCI Inbound Windows**

The PCI controller has three 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 PCI programming model. Updating one automatically updates the other. There is no default inbound window; if a PCI address does not match one of the inbound windows, this processor does not respond with an assertion of $\overline{\text{PCI_DEVSEL}}$. See [Section 23.4.6, “PCI Inbound Address Translation,”](#) for a detailed description of the PCI inbound windows.

6.2.10 has **Internal Memory Map**

All of the memory mapped configuration, control, and status registers in the device are contained within a 2-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 6.2.4.1, “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.

6.2.11 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.

6.3 System Configuration

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

6.3.1 System Configuration Register Memory Map

Table 6-20 shows the memory map for the system configuration registers.

Table 6-20. System Configuration Register Memory Map

Local Memory Offset (Hex)	Register	Access	Reset	Section/Page
System Configuration—Block Base Address 0x0_0000				
0x00100	System general purpose register low (SGPRL)	R/W	0x480E_FF20 or 0x0000_0000	6.3.2.1/6-17
0x00104	System general purpose register high (SGPRH)	R/W	0x0000_0000	6.3.2.2/6-18
0x00108	System part and revision ID register (SPRIDR)	R	0x8110_0010	6.3.2.3/6-18
0x0010C	Reserved	—	—	—
0x00110	System priority configuration register (SPCR)	R/W	0x0000_0000	6.3.2.4/6-19
0x00114	System I/O configuration register 1 (SICR_1)	R/W	0x0000_000A (other-boot) 0x0028_000A (esdhc-boot)	6.3.2.5/6-21
0x00118	System I/O configuration register 2 (SICR_2)	R/W	0x9040_5500 (PCIARB=1) 0x9042_5500 (PCIARB=0)	6.3.2.6/6-24
0x00128	DDR control driver register (DDRCDR)	R/W	0x0000_0000	6.3.2.9/6-29
0x0012C	DDR debug status register (DDRDSR)	R	0x3300_0000	6.3.2.10/6-30
0x00144	eSDHC Control Register (SDHCCR)	R/W	0x0000_0000	6.3.2.11/66-30

Table 6-20. System Configuration Register Memory Map (continued)

Local Memory Offset (Hex)	Register	Access	Reset	Section/Page
0x00148	CAN access control register (CAN_DBG_CTRL)	R/W	0x0000_0000	6.3.2.12/66-32
0x0014C	SPI chip select register (SPI_CS)	R/W	0x0000_0000	6.3.2.13/66-33
0x00150	General purpose register 1 (GPR_1)	R/W	0x0080_8001	6.3.2.14/66-34
0x00154–0x00167	Reserved	—	—	—
0x168	SIDCR2	R/W	0x0000_0000 (M66EN=0) 0xFC00_0000 (M66EN=1)	6.3.2.16/66-36
0x00169–0x001BC	Reserved	—	—	—
0x00154–0x00164	Reserved	—	—	—
0x00168	SIDCR2 ()			
0x0016C–0x001BC	Reserved	—	—	—
0x001C0	CAN interrupt status register (CAN_INT_STAT)	R	0x0000_0000	6.3.2.16/66-36
0x001C4	DUART interrupt status register (DUART_INT_STAT)	R	0x0000_0000	6.3.2.17/66-38
0x001C8	GPIO interrupt status register (GPIO_INT_STAT)	R	0x0000_0000	6.3.2.18/66-39
0x001CC–0x001FC	Reserved	—	—	—

6.3.2 System Configuration Registers

This section discusses the system configuration registers.

6.3.2.1 System General Purpose Register Low (SGPRL)

The system general purpose register low (SGPRL), shown in [Figure 6-10](#), can be used by software for any purpose. The values set in this register have no effect on the internal hardware.


Figure 6-10. System General Purpose Register Low (SGPRL)

Table 6-21 defines the bit fields of SGPRL.

Table 6-21. SGPRL Bit Settings

Bits	Name	Description
0–31	GP	General purpose

6.3.2.2 System General Purpose Register High (SGPRH)

The system general purpose register high (SGPRH), shown in Figure 6-11, can be used by software for any purpose. The values set in this register have no effect on the internal hardware.

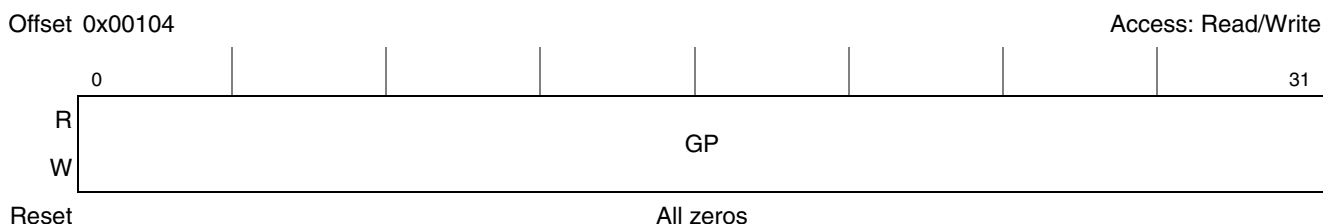


Figure 6-11. System General Purpose Register High (SGPRH)

Table 6-22 defines the bit fields of SGPRH.

Table 6-22. SGPRH Bit Settings

Bits	Name	Description
0–31	GP	General purpose

6.3.2.3 System Part and Revision ID Register (SPRIDR)

SPRIDR, shown in Figure 6-12, provides information about the device and revision numbers.

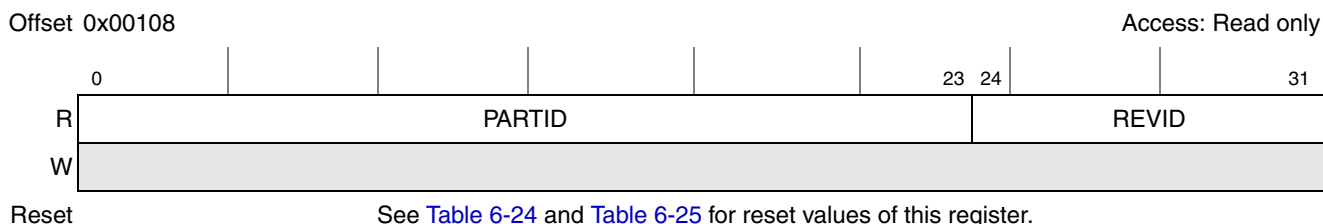


Figure 6-12. System Part and Revision ID Register (SPRIDR)

Table 6-23 defines the bit fields of SPRIDR.

Table 6-23. SPRIDR Bit Settings

Bits	Name	Description
0–23	PARTID	Part identification. This read-only field is mask-programmed with a code corresponding to the device number. It is intended to help factory test and user code that is sensitive to device changes. The device number changes according to manufacturing considerations. See Table 6-24 for values of this field.
24–31	REVID	Revision identification. This read-only field is mask-programmed with a code corresponding to the revision number of the part defined in PARTID field. It is intended to help factory test and user code that is sensitive to device changes. The mask number is programmed in a commonly changed layer, and changes with each mask set change. See Table 6-25 for values of this field.

6.3.2.3.1 SPRIDR[PARTID] Coding

Table 6-24 defines the reset values of SPRIDR[PARTID].

Table 6-24. PARTID Coding

PARTID	Device Name	Package Type
0x811000	MPC8309	PBGA

6.3.2.3.2 SPRIDR[REVID] Coding

Table 6-25 defines the reset values of SPRIDR[REVID].

Table 6-25. REVID Coding

REVID	Device Revision
0x10	1.0

6.3.2.4 System Priority and Configuration Register (SPCR)

The system priority and configuration register (SPCR), shown in Figure 6-13, controls the priority of requests for transactions on the internal system bus. This priority is considered by the system arbiter

System Configuration

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

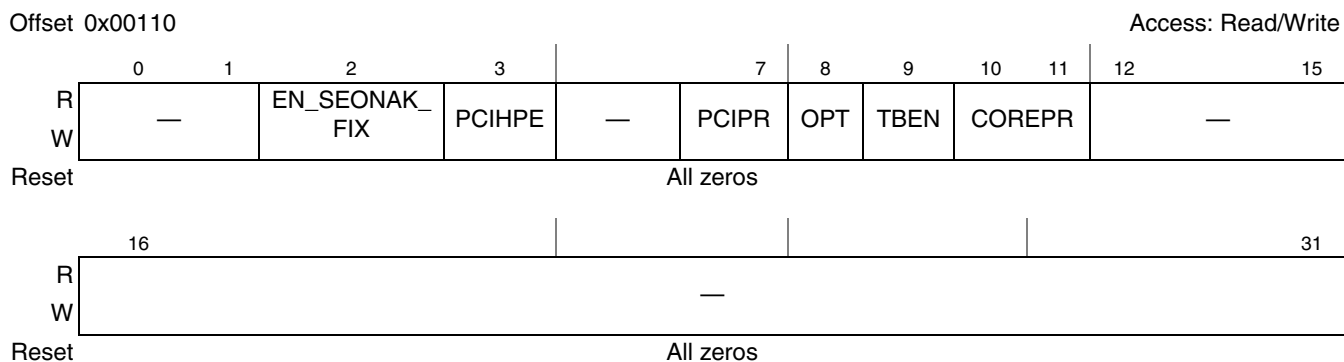


Figure 6-13. System Priority Configuration Register (SPCR)

Table 6-26 defines the bit fields of SPCR.

Table 6-26. SPCR Bit Settings

Bits	Name	Description
0–1	—	Reserved. Should be cleared.
2	EN_SEONAK_FIX	enable_se0nak_fix for USBDR If set, disables the SOF reporting when in SE0_NAK test mode.
3	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–	—	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) Note: DMA has the same priority as PCI.
8	OPT	Optimize. Setting this bit may enhance the performance of transactions issued to the internal coherent system bus (CSB) by the and the . 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 transactions sent to the internal CSB are not accessing devices in which speculative reads may change the state of the device (for example, FIFOs in which reading a byte may advance some internal counter). 0 No performance enhancement. 1 Performance enhancement by speculative reading is enabled.
9	TBEN	e300 core time base unit enable 0 Time base unit is disabled. 1 Time base unit is enabled.

Table 6-26. SPCR Bit Settings (continued)

Bits	Name	Description
10–11	COREPR	e300 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)
12–31	—	Reserved. Should be cleared.

6.3.2.5 System I/O Configuration Register 1 (SICR_1)

The system I/O configuration register 1 (SICR_1) controls the multiplexing of some of the device I/O pins. Each bit or set of bits in the SICR_1 selects which function is used by a certain group of the device pins.

Figure 6-14 shows SICR_1.

Offset 0x00114

Access: Read/Write

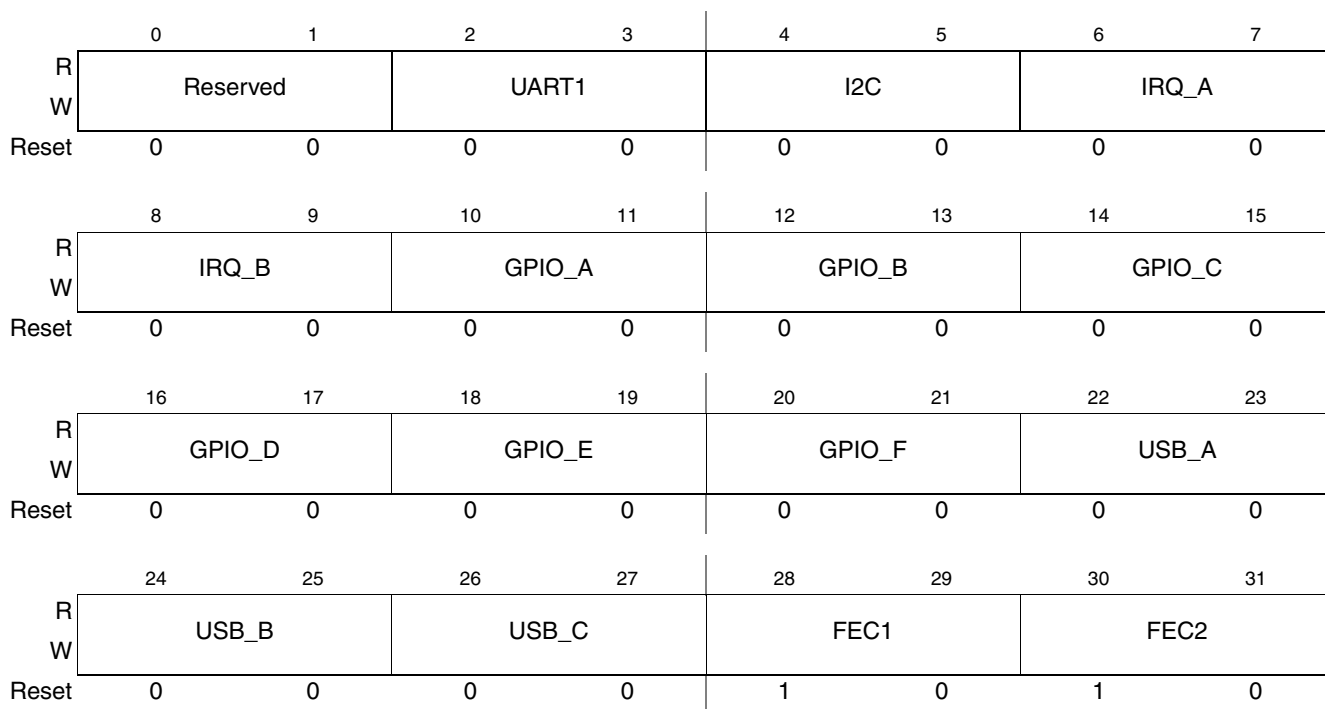


Figure 6-14. System I/O Configuration Register 1 (SICR_1)

NOTE

In case of eSDHC boot, the reset for SICR_1 changes from 0000_000A to 0028_000A.

Table 6-27 defines the bit fields of SICR_1 for MPC8309.

Table 6-27. SICR_1 Bit Settings for MPC8309

SICR_1[Bits] Value		0b00	0b01	0b10	0b11	Reset
Bits	Group	Pin Function 0	Pin Function 1	Pin Function 2	Pin Function 3	
0–1	—	—	—	—	—	00
2–3	UART[1]	UART1_SOUT[2]	$\overline{\text{UART1_RTS}}[1]$	—	—	00
		UART1_SIN[2]	$\overline{\text{UART1_CTS}}[1]$	—	—	
4–5	I2C	I2C_SDA[2]	$\overline{\text{CKSTOP_OUT}}$	—	—	00
		I2C_SCL[2]	$\overline{\text{CKSTOP_IN}}$	—	—	
6–7	IRQ_A	$\overline{\text{IRQ}}[1]$	$\overline{\text{MCP_OUT}}$	—	—	00
8–9	IRQ_B	$\overline{\text{IRQ}}[2]$	$\overline{\text{CKSTOP_IN}}$	—	—	00
		$\overline{\text{IRQ}}[3]$	$\overline{\text{CKSTOP_OUT}}$	$\overline{\text{INTA}}$	—	
10–11	GPIO_A	GPIO[0]	—	SD_CLK	MSRCID[0] (DDR ID)	00 (10 in case of eSDHC boot)
		GPIO[1]	—	SD_CMD	MSRCID[1] (DDR ID)	
		GPIO[2]	—	SD_CD	MSRCID[2] (DDR ID)	
		GPIO[3]	—	SD_WP	MSRCID[3] (DDR ID)	
		GPIO[4]	—	SD_DAT[0]	MSRCID[4] (DDR ID)	
		GPIO[5]	—	SD_DAT[1]	MDVAL(DDR ID)	
12–13	GPIO_B	GPIO[6]	—	SD_DAT[2]	QE_EXT_REQ_3	00 (10 in case of eSDHC boot)
		GPIO[7]	—	SD_DAT[3]	QE_EXT_REQ_1	
14–15	GPIO_C	GPIO[8]	RXCAN[1]	LSRCID[0] (DDR ID)	$\overline{\text{LCS}}[4]$	00
		GPIO[9]	TXCAN[1]	LSRCID[1] (DDR ID)	$\overline{\text{LCS}}[5]$	
16–17	GPIO_D	GPIO[10]	RXCAN[2]	LSRCID[2] (DDR ID)	$\overline{\text{LCS}}[6]$	00
		GPIO[11]	TXCAN[2]	LSRCID[3] (DDR ID)	$\overline{\text{LCS}}[7]$	
18–19	GPIO_E	GPIO[12]	RXCAN[3]	LSRCID[4] (DDR ID)	LCLK[1]	00
		GPIO[13]	TXCAN[3]	LDVAL(DDR ID)		

Table 6-27. SICR_1 Bit Settings for MPC8309 (continued)

SICR_1[Bits] Value		0b00	0b01	0b10	0b11	Reset
Bits	Group	Pin Function 0	Pin Function 1	Pin Function 2	Pin Function 3	
20–21	GPIO_F	GPIO[14]	RXCAN[4]	$\overline{\text{CKSTOP_OUT}}$	—	00
		GPIO[15]	TXCAN[4]	$\overline{\text{CKSTOP_IN}}$	—	
22–23	USB_A	USBDNR_NXT	UART2_SIN[1]	—	QE_EXT_REQ_4	00
		USBDNR_PCTL[0]	UART2_SOUT[1]	—	—	
24–25	USB_B	USBDNR_CLK	UART2_SIN[2]	$\overline{\text{UART2_CTS}}$	—	00
		USBDNR_PCTL[1]	UART2_SOUT[2]	$\overline{\text{UART2_RTS}}$	—	
26–27	USB_C	USBDNR_DIR	—	—	—	00
		USBDNR_PWRFLT	—	—	—	
		USBDNR_STP	—	—	QE_EXT_REQ_2	
28–29	FEC1	FEC1_COL	GTM1_TIN[1]	GPIO[16]	—	10
		FEC1_CRS	$\overline{\text{GTM1_TGATE}}[1]$	GPIO[17]	—	
		FEC1_RX_CLK (CLK9)	—	GPIO[18]	—	
		FEC1_RX_DV ¹	GTM1_TIN[2]	GPIO[19]	—	
		FEC1_RX_ER ¹	$\overline{\text{GTM1_TGATE}}[2]$	GPIO[20]	—	
		FEC1_RXD[0] ¹	—	GPIO[21]	—	
		FEC1_RXD[1]	GTM1_TIN[3]	GPIO[22]	—	
		FEC1_RXD[2]	$\overline{\text{GTM1_TGATE}}[3]$	GPIO[23]	—	
		FEC1_RXD[3]	—	GPIO[24]	—	
		FEC1_TX_CLK (CLK10)	GTM1_TIN[4]	GPIO[25]	—	
		FEC1_TX_EN ¹	$\overline{\text{GTM1_TGATE}}[4]$	GPIO[26]	—	
		FEC1_TX_ER	$\overline{\text{GTM1_TOUT}}[4]$	GPIO[27]	—	
		FEC1_TXD[0] ¹	$\overline{\text{GTM1_TOUT}}[1]$	GPIO[28]	—	
		FEC1_TXD[1]	$\overline{\text{GTM1_TOUT}}[2]$	GPIO[29]	—	
		FEC1_TXD[2]	$\overline{\text{GTM1_TOUT}}[3]$	GPIO[30]	—	
FEC1_TXD[3]	—	GPIO[31]	—			

Table 6-27. SICR_1 Bit Settings for MPC8309 (continued)

SICR_1[Bits] Value		0b00	0b01	0b10	0b11	Reset
Bits	Group	Pin Function 0	Pin Function 1	Pin Function 2	Pin Function 3	
30–31	FEC2	FEC2_COL	GTM2_TIN[1]	GPIO[32]	—	10
		FEC2_CRS	$\overline{\text{GTM2_TGATE}}[1]$	GPIO[33]	—	
		FEC2_RX_CLK (CLK7)	—	GPIO[34]	—	
		FEC2_RX_DV ¹	GTM2_TIN[2]	GPIO[35]	—	
		FEC2_RX_ER ¹	$\overline{\text{GTM2_TGATE}}[2]$	GPIO[36]	—	
		FEC2_RXD[0] ¹	—	GPIO[37]	—	
		FEC2_RXD[1]	GTM2_TIN[3]	GPIO[38]	—	
		FEC2_RXD[2]	$\overline{\text{GTM2_TGATE}}[3]$	GPIO[39]	—	
		FEC2_RXD[3]	—	GPIO[40]	—	
		FEC2_TX_CLK (CLK8)	GTM2_TIN[4]	GPIO[41]	—	
		FEC2_TX_EN ¹	$\overline{\text{GTM2_TGATE}}[4]$	GPIO[42]	—	
		FEC2_TX_ER	$\overline{\text{GTM2_TOUT}}[4]$	GPIO[43]	—	
		FEC2_TXD[0] ¹	$\overline{\text{GTM2_TOUT}}[1]$	GPIO[44]	—	
		FEC2_TXD[1]	$\overline{\text{GTM2_TOUT}}[2]$	GPIO[45]	—	
		FEC2_TXD[2]	$\overline{\text{GTM2_TOUT}}[3]$	GPIO[46]	—	
FEC2_TXD[3]	—	GPIO[47]	—			

¹ For multiplexed QE_UART signals, refer to [Table 6-29](#).

NOTE

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

6.3.2.6 System I/O Configuration Register 2 (SICR_2)

The system I/O configuration register 2, shown in [Figure 6-15](#), 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.

Offset 0x00118

Access: Read/Write

	0	1	2	3	4	5	6	7
R	FEC3		HDLC1_A		eLBC_A		eLBC_B	
W	FEC3		HDLC1_A		eLBC_A		eLBC_B	
Reset	1	0	0	1	0	0	0	0
	8			11	12	13	14	15
R	HDLC2_A		—		USB_D		PCI	
W	HDLC2_A		—		USB_D		PCI	
Reset	0	1	0	0	0	0	0	0
	16	17	18	19	20	21	22	23
R	HDLC1_B		HDLC1_C		HDLC2_B		HDLC2_C	
W	HDLC1_B		HDLC1_C		HDLC2_B		HDLC2_C	
Reset	0	1	0	1	0	1	0	1
	24	25	26	27	28	29	30	31
R	QIESCE_B		—					
W	QIESCE_B		—					
Reset	0	0	0	0	0	0	0	0

Figure 6-15. System I/O Configuration Register 2 (SICR_2)

NOTE

The reset value shown is for the case when PCIARB = 1. If PCIARB = 0, the reset for SICR_2 changes from 9040_5500 to 9042_5500.

Table 6-28 defines the bit fields of SICR_2. Each Pin Function column lists the name of the multi-function pin used in this option.

Table 6-28. SICR_2 Bit Settings for MPC8309

SICR_2[Bits] Value		0b00	0b01	0b10	0b11	Reset Value
Bits	Group	Pin Function 0	Pin Function 1	Pin Function 2	Pin Function 3	
0-1	FEC3	FEC3_COL	—	GPIO[48]	—	10
		FEC3_CRS	—	GPIO[49]	—	
		FEC3_RX_CLK (CLK11)	—	GPIO[50]	—	
		FEC3_RX_DV ¹	—	GPIO[51]	—	
		FEC3_RX_ER ¹	—	GPIO[52]	—	
		FEC3_RXD[0]	—	GPIO[53]	—	
		FEC3_RXD[1]	—	GPIO[54]	—	
		FEC3_RXD[2]	TSEC_TMR_TRIG[1]	GPIO[55]	—	
		FEC3_RXD[3] ¹	TSEC_TMR_TRIG[2]	GPIO[56]	—	
		FEC3_TX_CLK (CLK12)	TSEC_TMR_CLK	GPIO[57]	—	
		FEC3_TX_EN ¹	TSEC_TMR_GCLK	GPIO[58]	—	
		FEC3_TX_ER	TSEC_TMR_PP[1]	GPIO[59]	—	
		FEC3_TXD[0]	TSEC_TMR_PP[2]	GPIO[60]	—	
		FEC3_TXD[1]	TSEC_TMR_PP[3]	GPIO[61]	—	
		FEC3_TXD[2]	TSEC_TMR_ALARM[1]	GPIO[62]	—	
FEC3_TXD[3] ¹	TSEC_TMR_ALARM[2]	GPIO[63]	—			
2-3	HDLC1_A	HDLC1_TXD ¹	GPIO[2]	TDM1_TD	—	01
		HDLC1_RXD ¹	GPIO[3]	TDM1_RD	—	
		$\overline{\text{HDLC1_CD}}^1$	GPIO[4]	TDM1_TFS	—	
		$\overline{\text{HDLC1_CTS}}^1$	GPIO[5]	TDM1_RFS	—	
		$\overline{\text{HDLC1_RTS}}^1$	GPIO[6]	$\overline{\text{TDM1_STROBE}}$	—	
4-5	eLBC_A	LA[16]	—	—	—	00
6-7	eLBC_B	LCLK[0]	—	—	—	00
8-9	HDLC2_A	HDLC2_TXD ¹	GPIO[18]	TDM2_TD	—	01
		HDLC2_RXD ¹	GPIO[19]	TDM2_RD	—	
		$\overline{\text{HDLC2_CD}}^1$	GPIO[20]	TDM2_TFS	—	
		$\overline{\text{HDLC2_CTS}}^1$	GPIO[21]	TDM2_RFS	—	
		$\overline{\text{HDLC2_RTS}}^1$	GPIO[22]	$\overline{\text{TDM2_STROBE}}$	—	

Table 6-28. SICR_2 Bit Settings for MPC8309

SICR_2[Bits] Value		0b00	0b01	0b10	0b11	Reset Value
Bits	Group	Pin Function 0	Pin Function 1	Pin Function 2	Pin Function 3	
10–11	—	—	—	—	—	00
12–13	USB_D	USBDR_TXDRXD[0]	—	GPIO[32]		00
		USBDR_TXDRXD[1]	—	GPIO[33]		
		USBDR_TXDRXD[2]	—	GPIO[34]	QE_BRG_1	
		USBDR_TXDRXD[3]	—	GPIO[35]	QE_BRG_2	
		USBDR_TXDRXD[4]	—	GPIO[36]	QE_BRG_3	
		USBDR_TXDRXD[5]	—	GPIO[37]	QE_BRG_4	
		USBDR_TXDRXD[6]	—	GPIO[38]	QE_BRG_9	
		USBDR_TXDRXD[7]	—	GPIO[39]	QE_BRG_11	
14–15	PCI	$\overline{\text{PCI_REQ}}[1]$	—	CPCI_HS_ES	—	00 (10 if PCIA RB = 0)
		$\overline{\text{PCI_GNT}}[1]$	—	CPCI_HS_LED	—	
		$\overline{\text{PCI_GNT}}[2]$	—	CPCI_HS_ENUM	—	
16–17	HDLC1_B	HDLC1_TXCLK (CLK16)	GPIO[0]	QE_BRG_5	TDM1_TCK (CLK4)	01
18–19	HDLC1_C	HDLC1_RXCLK (CLK15)	GPIO[1]	TDM1_RCK (CLK3)	—	01
20–21	HDLC2_B	HDLC2_TXCLK (CLK14)	GPIO[16]	QE_BRG_7	TDM2_TCK (CLK6)	01
22–23	HDLC2_C	HDLC2_RXCLK (CLK13)	GPIO[17]	TDM2_RCK (CLK5)	QE_BRG_8	01
24–25	QUIESCE_B	$\overline{\text{QUIESCE}}$	—	—	—	00
26–31	—	—	—	—	—	00

¹ For multiplexed QE_UART signals, refer to [Table 6-29](#).

Table 6-29. QE UART Multiplexing Details

UART Signals	UCC signals				
	UCC1	UCC2	UCC3	UCC5	UCC7
CTS_B	FEC1_RX_DV	FEC2_RX_DV	FEC3_RX_DV	HDLC2_CTS_B	HDLC1_CTS_B
CD_B	FEC1_RX_ER	FEC2_RX_ER	FEC3_RX_ER	HDLC2_CD_B	HDLC1_CD_B
SIN	FEC1_RXD0	FEC2_RXD0	FEC3_RXD0	HDLC2_RXD	HDLC1_RXD

Table 6-29. QE UART Multiplexing Details

UART Signals	UCC signals				
	UCC1	UCC2	UCC3	UCC5	UCC7
RTS_B	FEC1_TX_EN	FEC2_TX_EN	FEC3_TX_EN	HDLC2_RTS_B	HDLC1_RTS_B
SOUT	FEC1_TXD0	FEC2_TXD0	FEC3_TXD0	HDLC2_TXD	HDLC1_TXD

6.3.2.7 Selection of Pin Functions During Reset

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

Pin Name	Function selection during reset	Normal Operation Mode
USBDR_PCTL1	LB_POR_BOOT_ERR	USBDR_PCTL1
USBDR_PCTL0	LB_POR_CFG_BOOT_ECC	USBDR_PCTL0
HDLC1_TXD	CFG_RESET_SOURCE[0]	HDLC1_TXD
HDLC1_RTS	CFG_RESET_SOURCE[1]	HDLC1_RTS
HDLC2_TXD	CFG_RESET_SOURCE[2]	HDLC2_TXD
HDLC2_RTS	CFG_RESET_SOURCE[3]	HDLC2_RTS

6.3.2.8 Debug Configuration

Debug information may be driven on the device pins. This information can identify the internal source of a transaction that reached the DDR SDRAM or local bus interfaces. The device can be configured to drive the MSRCID[0:4] and MDVAL, 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 7.2.6, “Arbiter Event Attributes Register \(AEATR\)”](#)).

6.3.2.9 DDR Control Driver Register (DDRCDR)

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

DDRCDR is shown in [Figure 6-16](#).

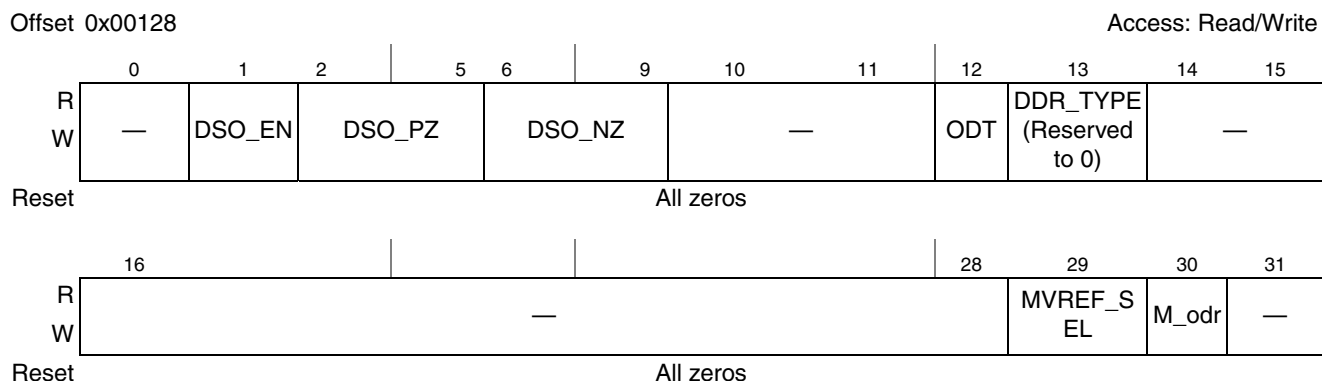


Figure 6-16. DDR Control Driver Register (DDRCDR)

[Table 6-30](#) shows the bit definition of the DDRCDR.

Table 6-30. DDRCDR Field Descriptions

Bits	Name	Description
0	—	Reserved
1	DSO_EN	0 DDR driver software override disable 1 DDR driver software override enable
2–5	DSO_PZ	DDR driver software p-impedance override 0000 Half strength—Highest Z 1000 Much higher Z than nominal 1100 Higher Z than nominal 1110 Nominal impedance setting 1111 Lower Z than nominal
6–9	DSO_NZ	DDR driver software n-impedance override 0000 Half strength—Highest Z 1000 Much higher Z than nominal 1100 Higher Z than nominal 1110 Nominal impedance setting 1111 Lower Z than nominal
10–11	—	Reserved. Should be cleared.
12	ODT	ODT termination value for I/Os 0 75 Ω 1 150 Ω
13	DDR_TYPE	Selects voltage level for DDR pads 0 DDR2 (1.8V mode) nominal impedance—18 Ω Note: DDR_TYPE must be set according to the logical type of the DDR memory devices, as it effects logic behavior of the DDR controller as well as the physical parameters of the DDR I/O pads.
14–28	—	Reserved

Table 6-30. DDRCDR Field Descriptions (continued)

Bits	Name	Description
29	MVREF_SEL	MVREF_SEL 0 MVREF is i/p from external source 1 MVREF is generated internally from GVDD
30	M_odr	Disable memory transaction reordering 0 Memory transaction reordering enabled 1 Memory transaction reordering disabled
31	—	Reserved

6.3.2.10 DDR Debug Status Register (DDRDSR)

Figure 6-17 contains the debug status bits from the DDR SDRAM controller.

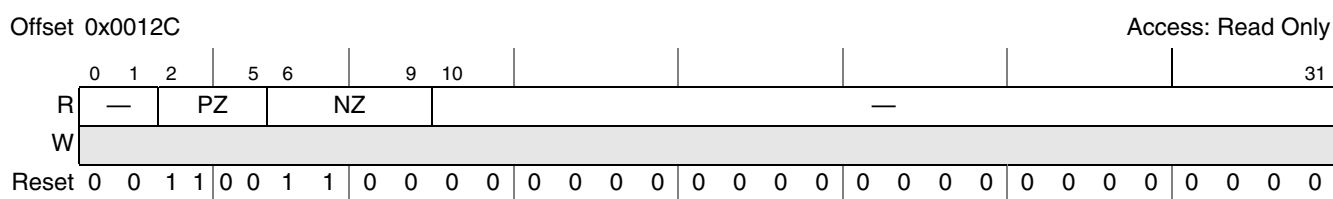


Figure 6-17. DDR Debug Status Register (DDRDSR)

Table 6-31 shows the bit settings of the DDRDSR.

Table 6-31. DDRDSR Field Descriptions

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

6.3.2.11 eSDHC Control Register (SDHCCR)

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

Figure 6-18 shows the SDHCCR bit settings.

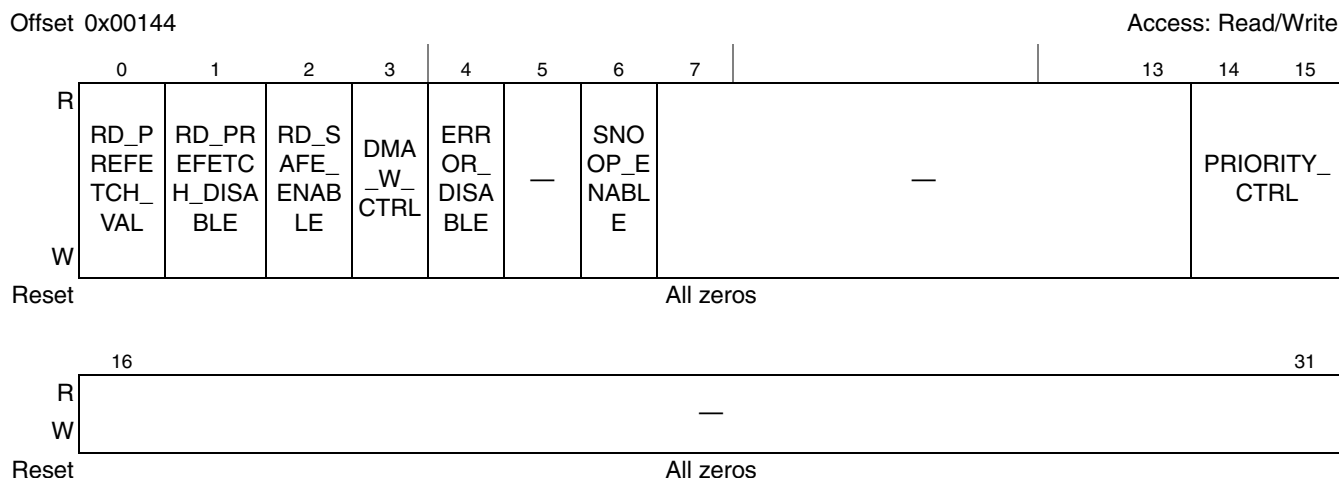


Figure 6-18. eSDHC Control Register (SDHCCR)

Table 6-32 describes the bits of the SDHCCR register.

Table 6-32. SDHCCR Field Description

Bits	Name	Description
0	RD_PREFETCH_VAL	This determines the prefetch byte count to be used if RD_PREFETCH_DISABLE is not set. 0 32 byte prefetch 1 64 byte prefetch
1	RD_PREFETCH_DISABLE	Read prefetch disable. This should be cleared if the target of read DMA operation is a well behaved memory which is not affected by the read operation and returns the same data if read again from the same location. This means that prefetch of data can be done by the internal bus units and it results in faster read completion. 0 It is allowed to prefetch data on DMA read operation 1 It is not allowed to prefetch data on DMA read operation
2	RD_SAFE_ENABLE	Read Safe enable. This bit should be set only if the target of read DMA operation is a well behaved memory which is not affected by the read operation and returns the same data if read again from the same location. This means that unaligned reading operation can be rounded up to enable more efficient read operations. 0 It is not safe to read more bytes that were intended 1 It is safe to read more bytes that were intended
3	DMA_W_CTRL	Write delay control bit. 0 No delay 1 Delay the Write/Read transaction till the interrupt is active
4	ERROR_DISABLE	Ignore or react to bus errors. 0 React to bus transaction errors 1 Ignore bus transaction errors
5	—	Reserved.
6	SNOOP_ENABLE	Snoop attribute. 0 DMA transactions are not snooped by e300 CPU data cache 1 DMA transactions are snooped by e300 CPU data cache
7–13	Reserved	—

Table 6-32. SDHCCR Field Description (continued)

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

6.3.2.12 CAN Access Control Register (CAN_DBG_CTRL)

The CAN access control register, shown in [Figure 6-19](#), contains the bits to control the FlexCAN module. The CAN module can be put to the debug mode by setting the DBG_EN bit. For more information on how to use the FlexCAN module after it is put into the debug mode, see FlexCAN chapter.

FlexCAN module memory map can be set into the supervisor mode by setting the SUP_EN bit.

Offset 0x00148

Access: Read/Write

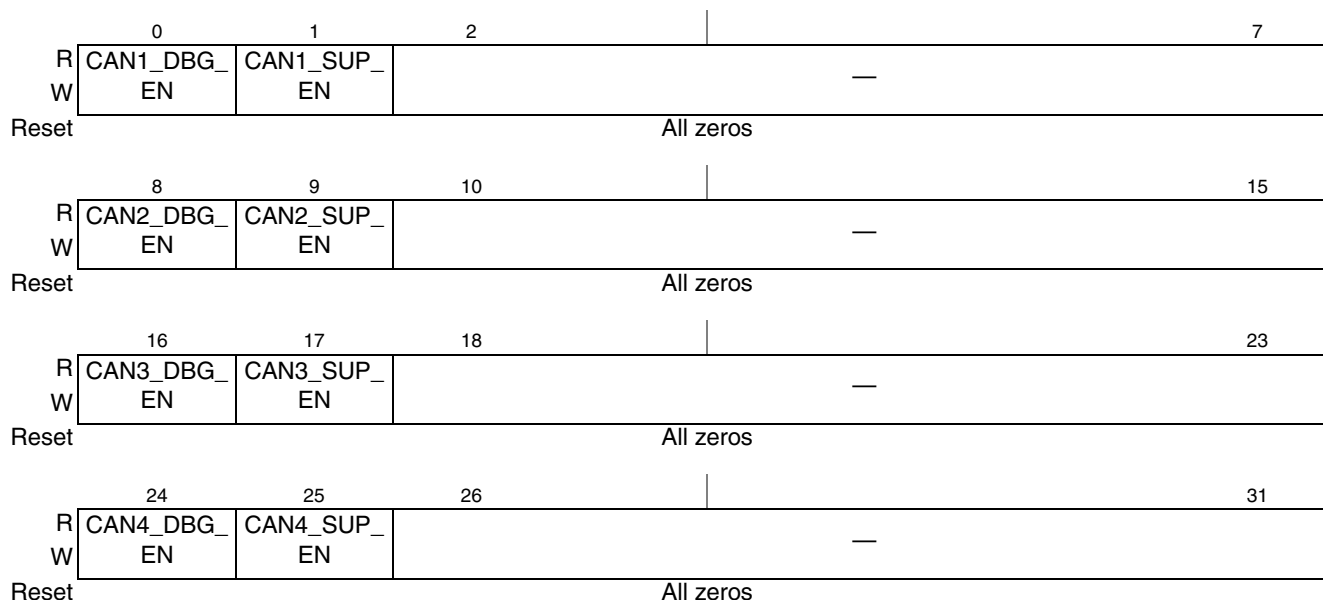


Figure 6-19. CAN Access Control Register (CAN_DBG_CTRL)

Table 6-33 defines the bit fields of CAN_DBG_CTRL.

Table 6-33. CAN_DBG_CTRL Field Descriptions

Bits	Name	Description
0	CAN1_DBG_EN	0 CAN1 is in normal mode 1 CAN1 is in debug mode
1	CAN1_SUP_EN	0 CAN1 is in normal mode 1 CAN1 is in supervisor mode
2–7	—	Reserved
8	CAN2_DBG_EN	0 CAN2 is in normal mode 1 CAN2 is in debug mode
9	CAN2_SUP_EN	0 CAN2 is in normal mode 1 CAN2 is in supervisor mode
10–15	—	Reserved
16	CAN3_DBG_EN	0 CAN3 is in normal mode 1 CAN3 is in debug mode
17	CAN3_SUP_EN	0 CAN3 is in normal mode 1 CAN3 is in supervisor mode
18–23	—	Reserved
24	CAN4_DBG_EN	0 CAN4 is in normal mode 1 CAN4 is in debug mode
25	CAN4_SUP_EN	0 CAN4 is in normal mode 1 CAN4 is in supervisor mode
26–31	—	Reserved

6.3.2.13 SPI Chip Select Register (SPI_CS)

The SPI chip select register, shown in [Figure 6-20](#), is used to control $\overline{\text{SPISEL_BOOT}}$ pin. Any value written to this register is reflected on $\overline{\text{SPISEL_BOOT}}$.

Offset 0x0014C

Access: Read/Write



Figure 6-20. SPI Chip Select Register (SPI_CS)

Table 6-35 defines the bit fields of SPI_CS.

Table 6-34. SPI_CS Field Descriptions

Bits	Name	Description
0	SPI_BOOT_SEL	Any value written to it is reflected on $\overline{\text{SPISEL_BOOT}}$. 0 $\overline{\text{SPISEL_BOOT}}$ goes low 1 $\overline{\text{SPISEL_BOOT}}$ goes high
1–31	—	Reserved

6.3.2.14 General Purpose Register 1 (GPR_1)

The general purpose register 1, shown in Figure 6-21, contains the control bit for PULLUP_EN for a set of pins, HDLC open drain selection, QUICC Engine PIO (programmable IO) selection, QUICC Engine debug enable, and Ethernet management interface selection..

Offset 0x00150

Access: Read/Write

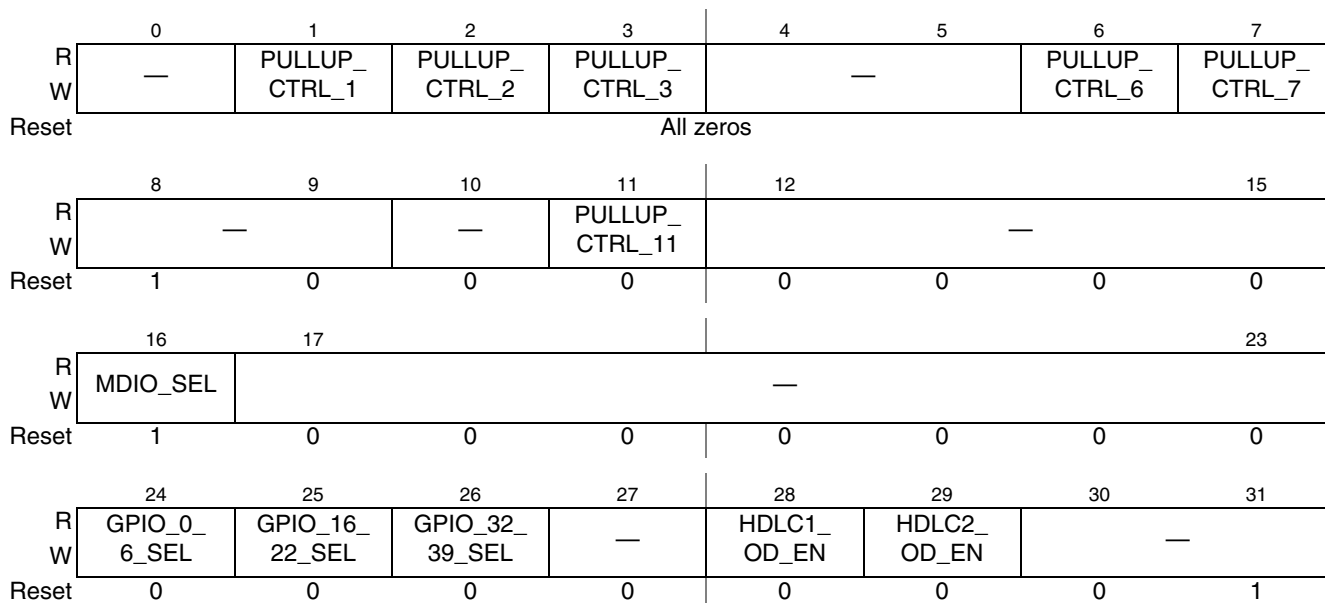


Figure 6-21. General Purpose Register 1 (GPR_1)

Table 6-35 defines the bit fields of GPR_1.

Table 6-35. GPR_1 Field Descriptions

Bits	Name	Description
0	—	Reserved
1	PULLUP_CTRL_1	1 Pull-up disable 0 Pull-up enable for the following pads: FEC1_COL, FEC1_CRS, FEC1_RX_CLK, FEC1_RX_DV, FEC1_RX_ER, FEC1_RXD[0], FEC1_RXD[1], FEC1_RXD[2], FEC1_RXD[3], FEC1_TX_CLK, FEC1_TX_EN, FEC1_TX_ER, FEC1_TXD[0], FEC1_TXD[1], FEC1_TXD[2], FEC1_TXD[3]
2	PULLUP_CTRL_2	1 Pull-up disable 0 Pull-up enable for the following pads: FEC2_COL, FEC2_CRS, FEC2_RX_CLK, FEC2_RX_DV, FEC2_RX_ER, FEC2_RXD[0], FEC2_RXD[1], FEC2_RXD[2], FEC2_RXD[3], FEC2_TX_CLK, FEC2_TX_EN, FEC2_TX_ER, FEC2_TXD[0], FEC2_TXD[1], FEC2_TXD[2], FEC2_TXD[3]
3	PULLUP_CTRL_3	1 Pull-up disable 0 Pull-up enable for the following pads: FEC3_COL, FEC3_CRS, FEC3_RXD[1], FEC3_RX_DV, FEC3_TXD[1], FEC3_TXD[0], FEC3_RXD[0], FEC3_RXD[3], FEC3_RX_CLK, FEC3_TX_EN, FEC3_TX_ER, FEC3_TXD[3], FEC3_TXD[2], FEC3_TX_CLK, FEC3_RXD[2], FEC3_RX_ER
4–5	—	Reserved

Table 6-35. GPR_1 Field Descriptions (continued)

Bits	Name	Description
6	PULLUP_CTRL_6	1 Pull-up disable 0 Pull-up enable for the following pads: GPIO[0], GPIO[1], GPIO[2], GPIO[3], GPIO[4], GPIO[5], GPIO[6], GPIO[7]
7	PULLUP_CTRL_7	1 Pull-up disable 0 Pull-up enable for the following pads: GPIO[8], GPIO[9], GPIO[10], GPIO[11], GPIO[12], GPIO[13], GPIO[14], GPIO[15]
8–10	—	Reserved
11	PULLUP_CTRL_11	1 Pull-up disable 0 Pull-up enable for the following pads: HDLC1_CD, HDLC1_CTS, HDLC1_RTS, HDLC1_TXCLK, HDLC1_TXD, HDLC1_RXD, HDLC1_RX_CLK, HDLC2_CD, HDLC2_CTS, HDLC2_RTS, HDLC2_RXCLK, HDLC2_RXD, HDLC2_TXCLK, HDLC2_TXD
12–15	—	Reserved
16	MDIO_SEL	The pins FEC_MDC and FEC_MDIO forms the Ethernet management port. This can be controlled from SPI (SPI inside QUICC Engine) for faster operation or via the UCC based MDC/MDIO. The selection is done by this bit. 0 QUICC Engine SPI selected as Ethernet management port 1 QUICC Engine MDC/MDIO selected as Ethernet management port
17–23	—	Reserved
24	GPIO_0_6_SEL	GPIO[0:6] is multiplexed at two locations: as primary function and multiplexed with HDLC/TDM pads. 0 Pads GPIO[0:6] selected 1 GPIO multiplexed with HDLC/TDM pads selected
25	GPIO_16_22_SEL	GPIO[16:22] is multiplexed at two locations: with FEC1 and with HDLC2. 0 GPIO multiplexed with FEC1 is selected 1 GPIO multiplexed with HDLC2 is selected
26	GPIO_32_39_SEL	GPIO[32:39] is multiplexed at two locations: with FEC2 and with USB. 0 GPIO multiplexed with FEC2 is selected 1 GPIO multiplexed with USB is selected
27	—	Reserved
28	HDLC1_OD_EN	0 HDLC1 normal mode 1 HDLC1 bus mode enable
29	HDLC2_OD_EN	0 HDLC2 normal mode 1 HDLC2 bus mode enable
30–31	—	Reserved

6.3.2.15 SIDCR2

168

6.3.2.16 CAN Interrupt Status Register (CAN_INT_STAT)

The CAN interrupt status register, shown in [Figure 6-22](#), indicates the interrupt status of the FlexCAN modules (CAN1, CAN2, CAN3, and CAN4).

All the interrupts of the FlexCAN modules are ORed together and connected to IPIC. Once an interrupt is detected by the software, the CAN_INT_STAT register is read to decode which CAN module has generated the interrupt. And then, the interrupt register is read from the corresponding module.

Offset 0x001C0

Access: Read Only

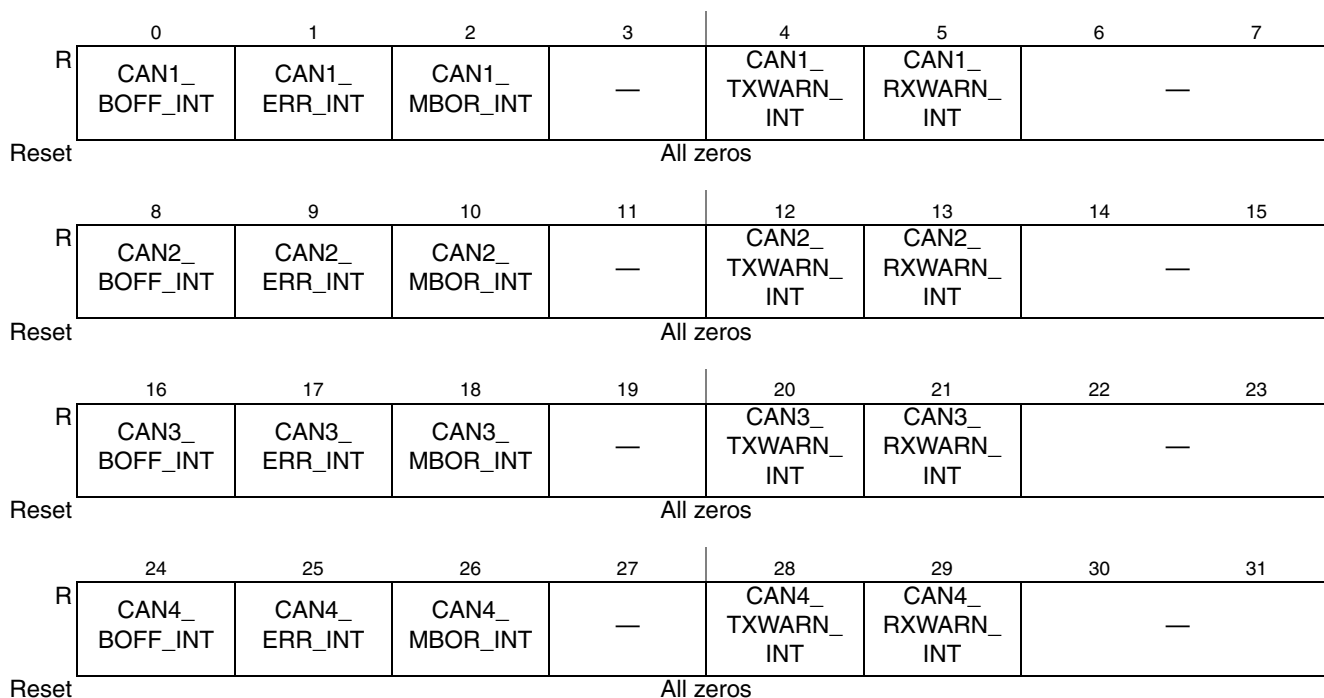


Figure 6-22. CAN Interrupt Status Register (CAN_INT_STAT)

[Table 6-36](#) defines the bit fields of CAN_INT_STAT. Refer to FlexCAN chapter for more information.

Table 6-36. CAN_INT_STAT Field Descriptions

Bits	Name	Description
0	CAN1_BOFF_INT	CAN1 bus Ored interrupt from MB. Refer to IFLAG1 and IFLAG 2 registers in FlexCAN module.
1	CAN1_ERR_INT	CAN1 error interrupt
2	CAN1_MBOR_INT	CAN1 MBOR interrupt
3	—	Reserved
4	CAN1_TXWARN_INT	CAN1 TX warning interrupt
5	CAN1_RXWARN_INT	CAN1 RX warning interrupt

Table 6-36. CAN_INT_STAT Field Descriptions (continued)

Bits	Name	Description
6–7	—	Reserved
8	CAN2_BOFF_INT	CAN2 bus off interrupt
9	CAN2_ERR_INT	CAN2 error interrupt
10	CAN2_MBOR_INT	CAN2 MBOR interrupt
11	—	Reserved
12	CAN2_TXWARN_INT	CAN2 TX warning interrupt
13	CAN2_RXWARN_INT	CAN2 RX warning interrupt
14–15	—	Reserved
16	CAN3_BOFF_INT	CAN3 bus off interrupt
17	CAN3_ERR_INT	CAN3 error interrupt
18	CAN3_MBOR_INT	CAN3 MBOR interrupt
19	—	Reserved
20	CAN3_TXWARN_INT	CAN3 TX warning interrupt
21	CAN3_RXWARN_INT	CAN3 RX warning interrupt
22–23	—	Reserved
24	CAN4_BOFF_INT	CAN4 bus off interrupt
25	CAN4_ERR_INT	CAN4 error interrupt
26	CAN4_MBOR_INT	CAN4 MBOR interrupt
27	—	Reserved
28	CAN4_TXWARN_INT	CAN4 TX warning interrupt
29	CAN4_RXWARN_INT	CAN4 RX warning interrupt
30–31	—	Reserved

6.3.2.17 DUART Interrupt Status Register (DUART_INT_STAT)

The DUART interrupt status register, shown in [Figure 6-23](#), indicates the interrupt status of the DUART modules (DUART1 and DUART2).

All the interrupts of the DUART modules are ORed together and connected to IPIC. Once an interrupt is detected by the software, the DUART_INT_STAT register is read to decode which DUART has generated the interrupt. And then, the interrupt register is read from the corresponding DUART.

Offset 0x001C4

Access: Read/Write

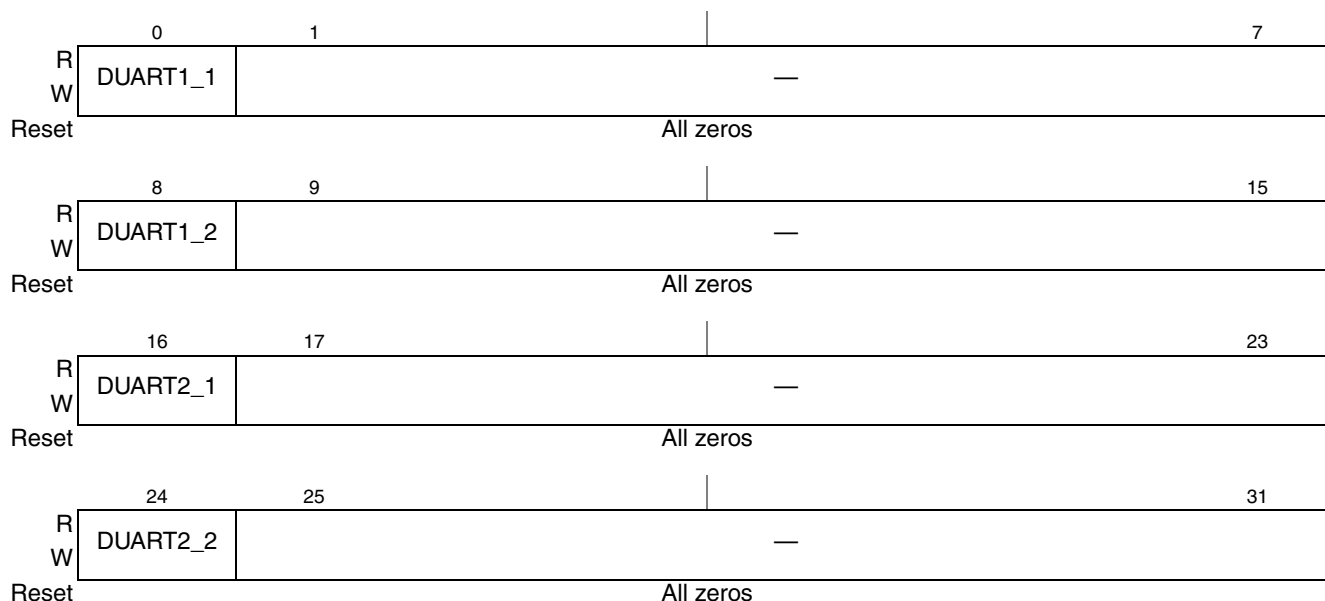


Figure 6-23. DUART Interrupt Status Register (DUART_INT_STAT)

[Table 6-37](#) defines the bit fields of DUART_INT_STAT.

Table 6-37. DUART_INT_STAT Field Descriptions

Bits	Name	Description
0	DUART1_1	0 No interrupt 1 Interrupt asserted
1–7	—	Reserved
8	DUART1_2	0 No interrupt 1 Interrupt asserted
9–15	—	Reserved
16	DUART2_1	0 No interrupt 1 Interrupt asserted
17–23	—	Reserved
24	DUART2_2	0 No interrupt 1 Interrupt asserted
25–31	—	Reserved

6.3.2.18 GPIO Interrupt Status Register (GPIO_INT_STAT)

The GPIO interrupt status register, shown in Figure 6-24, indicates the interrupt status of the GPIO modules (GPIO1 and GPIO2).

All the interrupts of the GPIO modules are ORed together and connected to IPIC. Once an interrupt is detected by the software, the GPIO_INT_STAT register is read to decode which GPIO module has generated the interrupt. And then, the interrupt register is read from the corresponding module.

Offset 0x001C8

Access: Read/Write

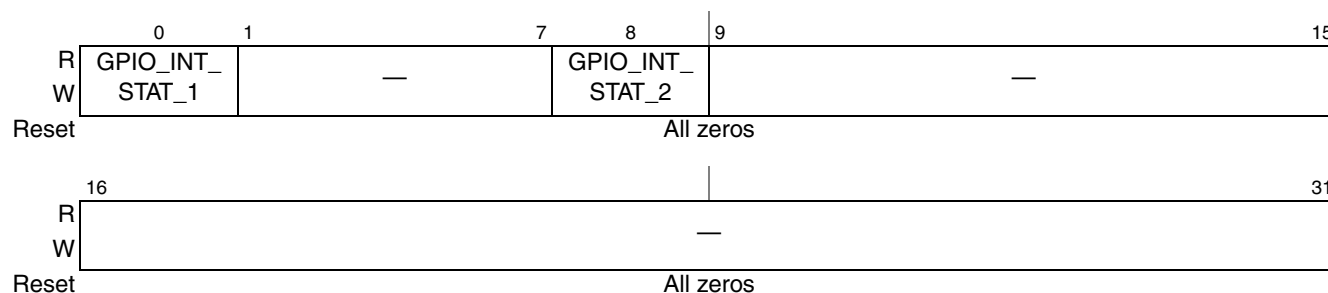


Figure 6-24. GPIO Interrupt Status Register (GPIO_INT_STAT)

Table 6-38 defines the bit fields of GPIO_INT_STAT.

Table 6-38. GPIO_INT_STAT Field Descriptions

Bits	Name	Description
0	GPIO_INT_STAT_1	GPIO 1 interrupt (GPIO[0:31]) 0 No interrupt 1 Interrupt asserted
1–7	—	Reserved
8	GPIO_INT_STAT_2	GPIO 2 interrupt (GPIO[32:63]) 0 No interrupt 1 Interrupt asserted
9–31	—	Reserved

6.3.3 Multisite Muxing

Multisite muxing allows user to choose an interface from more than one location.

For example, UART1 interface is provided at two locations: $\overline{\text{LCS}}[4:7]$ and USB. If the user chooses $\overline{\text{LCS}}[4:7]$, then UART1 cannot be used. Therefore, the same UART1 is also provided at another location, multiplexed with USB. However, note that, if the user wants to use both USB and $\overline{\text{LCS}}[4:7]$, UART1 cannot be used.

With MPC8309, the user has flexibility to choose the following interfaces from more than one location:

- $\overline{\text{CKSTOP_IN}}$ and $\overline{\text{CKSTOP_OUT}}$ are multiplexed at the following locations:
 - $\overline{\text{IRQ}}[2]$ and $\overline{\text{IRQ}}[3]$
 - IIC_SDA2 and IIC_SCL2

There is no control bit to select one of the two and any of the location can be selected using the SICR_1 and SICR_2 registers. For more information of SICR_1 and SICR_2 registers, see [Section 6.3.2.5, “System I/O Configuration Register 1 \(SICR_1\)”](#) and [Section 6.3.2.6, “System I/O Configuration Register 2 \(SICR_2\).”](#)

- GPIO[0:6], GPIO[16:22], and GPIO[32:39]
Refer to GPIO_0_6_SEL, GPIO_16_22_SEL, and GPIO_32_39_SEL bits in [Section 6.3.2.14, “General Purpose Register 1 \(GPR_1\).”](#)

6.4 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 serve. Additionally, the configuration, control, and status registers are also described. Note that individual chapters in this book describe specific initialization aspects for each individual block.

6.4.1 WDT Overview

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

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

[Figure 6-25](#) shows a high-level block diagram of the WDT.

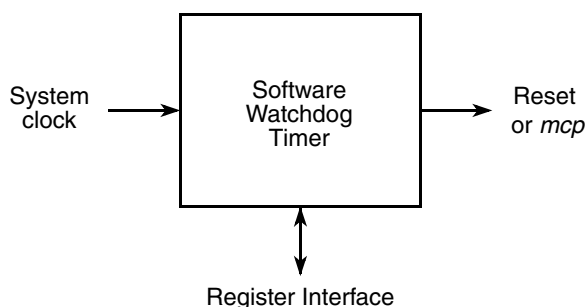


Figure 6-25. Software Watchdog Timer High-Level Block Diagram

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.

6.4.2 WDT Features

The WDT includes the following key features:

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

6.4.3 WDT Modes of Operation

The WDT unit can operate in the following modes:

- WDT enable/disable mode:
If the software watchdog timer is not needed, the user can disable it with software after a system reset. When the watchdog timer is disabled, the watchdog counter and prescaler counter are held in a stopped state.
- WDT output reset/interrupt mode:
Without software periodic servicing, the software watchdog timer times out and issues a reset or a nonmaskable interrupt (*mcp*)
- WDT prescaled/non-prescaled clock mode:
The WDT counter clock can be prescaled by programming the SWCRR[SWPR] bit, which controls the divide-by-65,536 of the WDT counter.

6.4.4 WDT Memory Map/Register Definition

The WDT programmable register map occupies 16 bytes of memory-mapped space. Reading undefined portions of the memory map returns all zeros; 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 6-39 shows the WDT memory map.

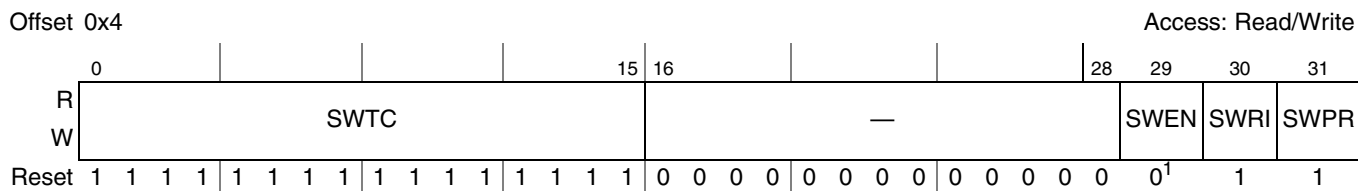
Table 6-39. WDT Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
Watchdog Timer (WDT)—Block Base Address 0x0_0200				
0x00–0x03	Reserved	—	—	—
0x004	System watchdog control register (SWCRR)	R/W	0xFFFF_0003 or 0xFFFF_0007 ¹	6.4.4.1/6-42
0x008	System watchdog count register (SWCNR)	R	0x0000_FFFF	6.4.4.2/6-43
0x00C–0x00D	Reserved	—	—	—
0x00E	System watchdog service register (SWSRR)	R/W	0x0000	6.4.4.3/6-43

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

6.4.4.1 System Watchdog Control Register (SWCRR)

The system watchdog control register (SWCRR), shown in [Figure 6-26](#), 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.



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

Figure 6-26. System Watchdog Control Register (SWCRR)

[Table 6-40](#) defines the bit fields of SWCRR.

Table 6-40. 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
29	SWEN	Watchdog enable bit Enables the watchdog timer. The reset value directly depends on the value of the RCWHR[SWEN] bit. It should be cleared by software after a system reset to disable the software watchdog timer. When the watchdog timer is disabled, the watchdog counter and prescaler counter are held in a stopped state. 0 Watchdog timer disabled 1 Watchdog timer enabled Note: After software writes the SWRI bit, the state of SWEN cannot be changed.
30	SWRI	Software watchdog reset/interrupt select bit A WDT time 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.

6.4.4.2 System Watchdog Count Register (SWCNR)

The system watchdog count register (SWCNR), shown in [Figure 6-27](#), 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.

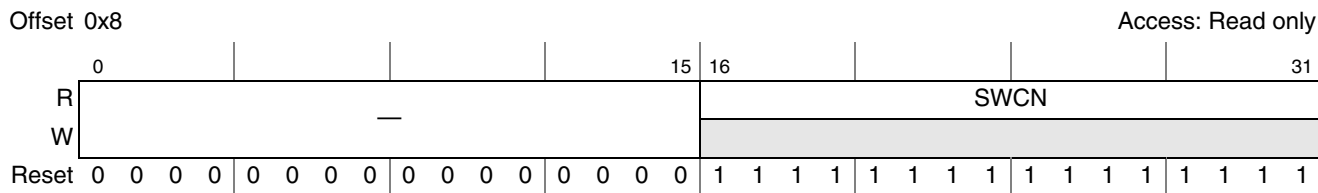


Figure 6-27. System Watchdog Count Register (SWCNR)

[Table 6-41](#) defines the bit fields of SWCNR.

Table 6-41. 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.

6.4.4.3 System Watchdog Service Register (SWSRR)

The system watchdog service register (SWSRR) is shown in [Figure 6-28](#). 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 6-28. System Watchdog Service Register (SWSRR)

Table 6-42 defines the bit fields of SWCNR.

Table 6-42. 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.

6.4.5 Functional Description

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

6.4.5.1 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 6-29 shows a state diagram for the watchdog timer.

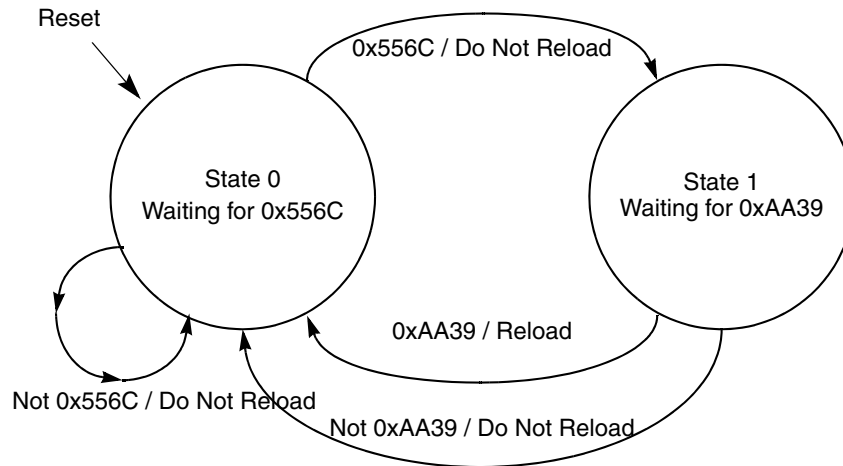


Figure 6-29. Software Watchdog Timer Service State Diagram

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 6-30 shows how to handle this need.

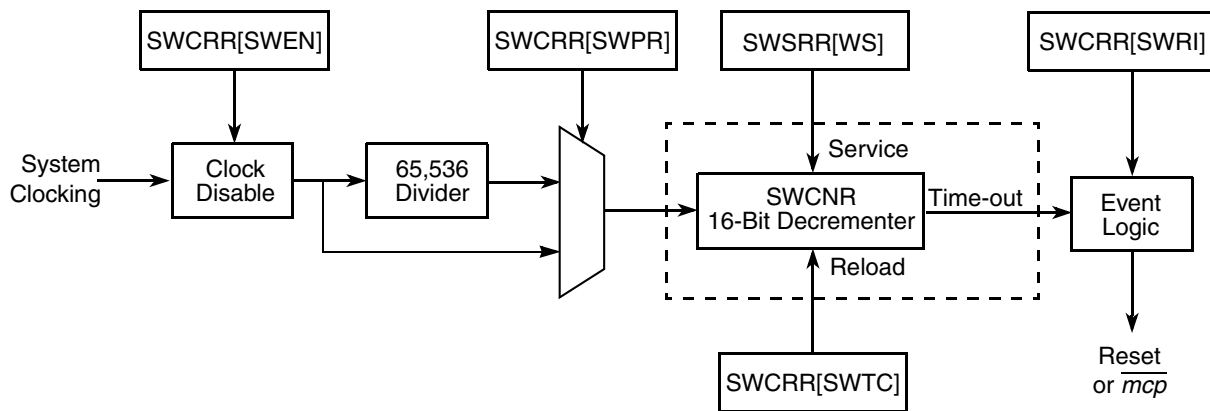


Figure 6-30. Software Watchdog Timer Functional Block Diagram

Figure 6-30 shows that the range is determined by SWCRR[SWTC]. The value in SWTC is then loaded into a 16-bit decremter clocked by the system clock. An additional divide-by-65,536 prescaler value is used when needed.

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

6.4.5.2 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 hard reset.
 - Selectable by RCWHR[8].
 - WDT disable mode (SWCRR[SWEN] = 0)
- WDT reset/interrupt output mode

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

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

 - Reset mode (SWCRR[SWRI] = 1).

Software watchdog timer causes a hard reset (this is the default value after hard reset).
 - Interrupt mode (SWCRR[SWRI] = 0).

Software watchdog timer causes a machine check interrupt to the core.
- WDT prescaled/non-prescaled clock mode

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

 - Prescale mode (SWCRR[SWPR] = 1)

The WDT clock is prescaled.
 - Non-prescale mode (SWCRR[SWPR] = 0)

The WDT clock is not prescaled.

6.4.6 Initialization/Application Information (WDT Programming Guidelines)

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

- WDT disabling

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

- WDT initial servicing

If the software watchdog timer is to be used, the special service sequence, described in [Section 6.4.5.1, “Software Watchdog Timer Unit,”](#) must be executed after system reset and not later than the first WDT time-out (~34.36 sec. for a 125-MHz system clock).

Subsequently, periodical WDT servicing should be performed according to the programming guidelines given in [Section 6.4.5.1, “Software Watchdog Timer Unit.”](#)

6.5 Real Time Clock Module (RTC)

This section describes 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.

6.5.1 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 6-31](#) shows the high level RTC block diagram.

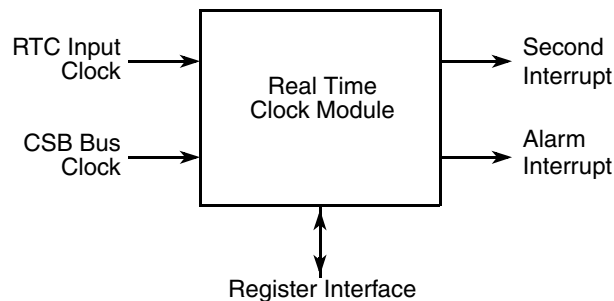


Figure 6-31. Real Time Clock Module High Level Block Diagram

6.5.2 Features

The key features of the RTC module 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

6.5.3 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

6.5.4 External Signal Description

Table 6-43 lists the external signals for the RTC module.

Table 6-43. RTC External Signals

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

6.5.5 RTC Memory Map/Register Definition

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

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

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

Table 6-39 shows the memory map of the RTC.

Table 6-44. RTC Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
Real Time Clock (RTC)—Block Base Address 0x0_0300				
0x00	Real time counter control register (RTCNR)	R/W	0x0000_0000	6.5.5.1/6-49
0x04	Real time counter load register (RTLDR)	R/W	0x0000_0000	6.5.5.2/6-49
0x08	Real time counter prescale register (RTPSR)	R/W	0x0000_0000	6.5.5.3/6-50
0x0C	Real time counter register (RTCTR)	R	0x0000_0000	6.5.5.4/6-50
0x10	Real time counter event register (RTEVR)	w1c	0x0000_0000	6.5.5.5/6-51
0x14	Real time counter alarm register (RTALR)	R/W	0xFFFF_FFFF	6.5.5.6/6-51
0x18–0x1F	Reserved	—	—	

6.5.5.1 Real Time Counter Control Register (RTCNR)

The real time counter control register (RTCNR), shown in Figure 6-32, is used to enable RTC functions. The register can be read at any time.

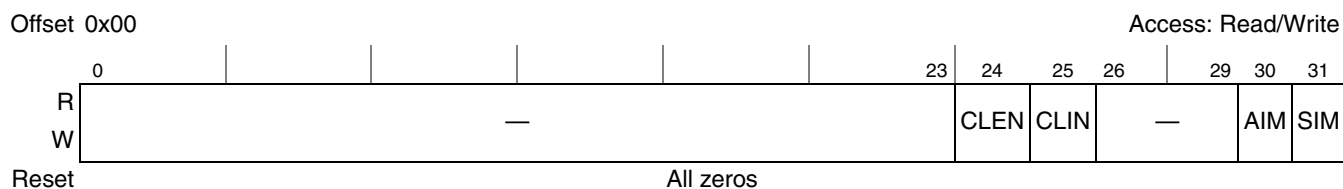


Figure 6-32. Real Time Counter Control Register (RTCNR)

Table 6-45 defines the bit fields of RTCNR.

Table 6-45. RTCNR Bit Settings

Bits	Name	Description
0–23	—	Write reserved, read = 0
24	CLEN	Clock enable control bit. This bit controls the counting of the RTC. When the RTC's clock is disabled, the counter maintains its old value. When the counter's clock is enabled, it continues counting using the previous value. 0 Disable counter. 1 Enable counter.
25	CLIN	Input clock control bit. The input clock to the RTC may be either the CSB clock or an external RTC clock. 0 The input clock to the RTC is CSB input clock. 1 The input clock to the RTC is the external RTC clock.
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[ALRM] 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.

6.5.5.2 Real Time Counter Load Register (RTLDR)

The real time counter load register (RTLDR), shown in Figure 6-33, contains the 32-bit value to be loaded in the 32-bit RTC counter.

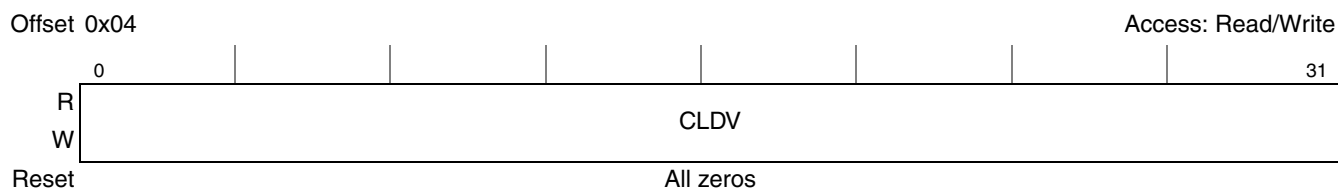


Figure 6-33. Real Time Counter Load Register (RTLDR)

Table 6-46 defines the bit fields of RTLDR.

Table 6-46. RTLDR Bit Settings

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

6.5.5.3 Real Time Counter Prescale Register (RTPSR)

The real time counter prescale register (RTPSR), shown in Figure 6-34, is a read/write register used to configure the RTC prescaler’s value.

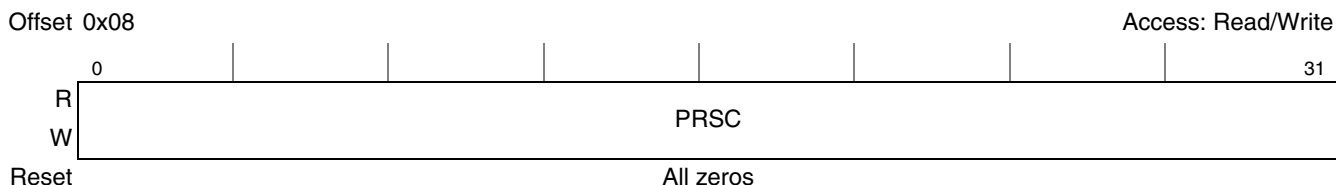


Figure 6-34. Real Time Counter Prescale Register (RTPSR)

Table 6-47 defines the bit fields of RTPSR.

Table 6-47. 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.

6.5.5.4 Real Time Counter Register (RTCTR)

The real time counter register (RTCTR), shown in Figure 6-35, 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.

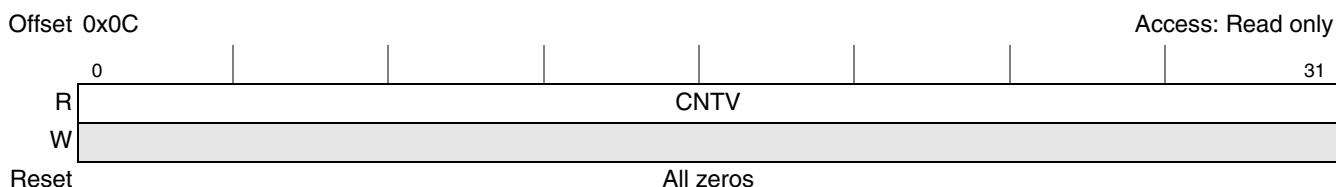


Figure 6-35. Real Time Counter Register (RTCTR)

Table 6-48 defines the bit fields of RTCTR.

Table 6-48. 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].

6.5.5.5 Real Time Counter Event Register (RTEVR)

The real time counter event register (RTEVR), shown in Figure 6-36, is used to report the source of the interrupts. The register can be read at any time.



Figure 6-36. 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 6-49 defines the bit fields of RTEVR.

Table 6-49. RTEVR Bit Settings

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

6.5.5.6 Real Time Counter Alarm Register (RTALR)

The real time counter alarm register (RTALR), shown in Figure 6-37, contains the 32-bit alarm (ALRM) value. When the value of the RTC counter equals the RTALR[ALRM] value, a maskable interrupt is generated.

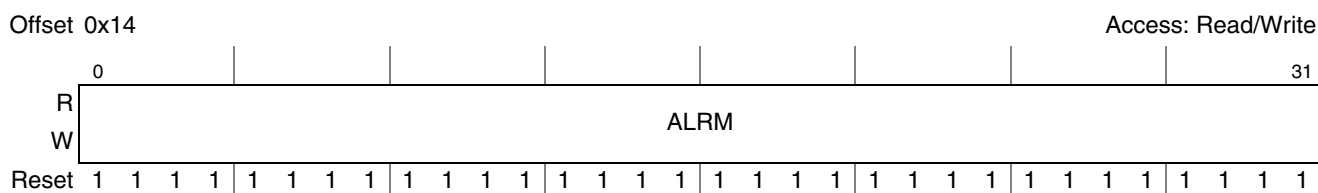


Figure 6-37. Real Time Counter Alarm Register (RTALR)

Table 6-50 defines the bit fields of RTALR.

Table 6-50. 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].

6.5.6 Functional Description

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

6.5.6.1 Real Time Counter Unit

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

The RTC can be initialized by software with an initial count value in the real time counter load register (RTLDR). It can also be programmed to generate an interrupt every second. The real time counter control register (RTCTR) is used to enable or disable various timer functions. The real time counter event register (RTEVR) is used to report the interrupt source. The RTC counter is reset to zero on hard reset or PORESET. RTC registers are initialized by the software. The RTC function can be disabled by programming the RTC registers.

Figure 6-38 shows the functional RTC block diagram.

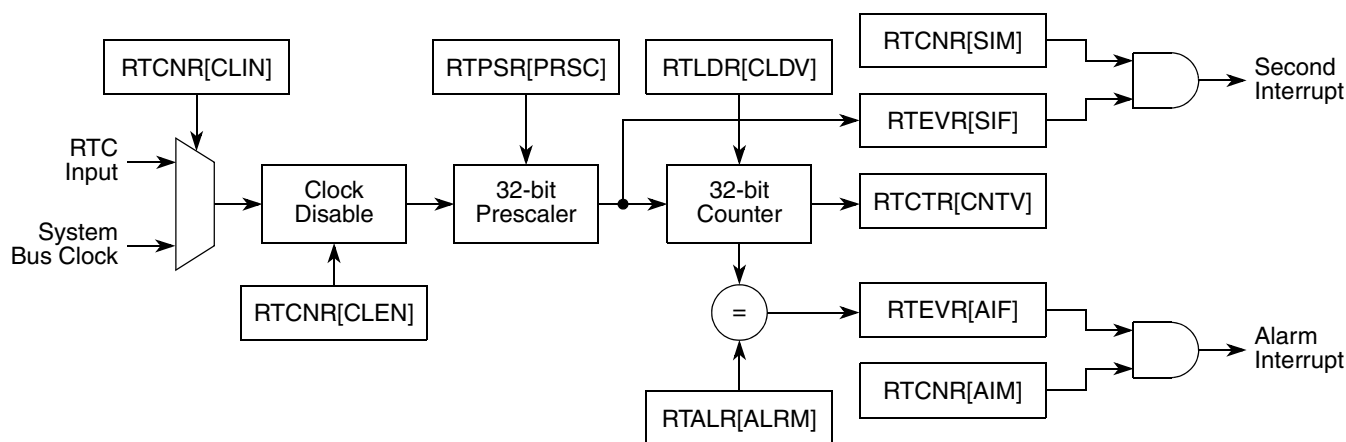


Figure 6-38. Real Time Clock Module Functional Block Diagram

6.5.6.2 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 an RTC 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, RTC sets the RTEVR[SIF] flag and generate an interrupt after the RTC's 32-bit counter reaches zero.
 - RTC every-second interrupt disable mode (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 RTC_PIT_CLK.
 - RTC uses the internal input clock mode (RTCNR[CLIN] = 0)
 - RTC uses the external RTC_PIT_CLK (RTCNR[CLIN] = 1)

6.5.7 RTC Initialization Sequence

The recommended initialization sequence for the RTC is as follows:

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

6.6 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.

6.6.1 PIT Overview

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

The PIT consists of a 32-bit down-counter which is decremented by a clock derived from a CSB clock or from an external 32.768-kHz crystal. The 32-bit counter decrements to zero when loaded with 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 6-39 shows the functional PIT block diagram.

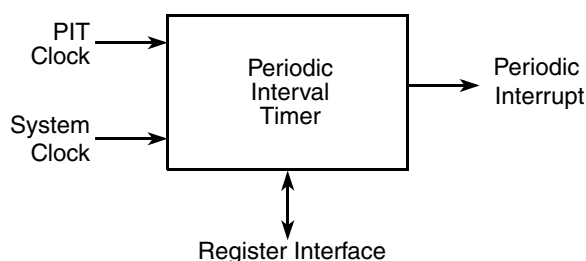


Figure 6-39. Periodic Interval Timer High Level Block Diagram

6.6.2 PIT Features

The key features of the PIT include the following:

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

6.6.3 PIT Modes of Operation

The PIT unit can operate in the following modes:

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

6.6.4 PIT External Signal Description

This section provides an overview and detailed descriptions of the PIT signals.

There is one distinct external input signal (PIT clock), defined in [Table 6-51](#).

Table 6-51. PIT Signal Properties

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

[Table 6-52](#) describes of the external PIT signal.

Table 6-52. PIT External Signal—Detailed Signal Descriptions

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

6.6.5 PIT Memory Map/Register Definition

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

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

All addresses used in this chapter are offsets from PIT base, as defined in [Chapter 2, “Memory Map.”](#)

[Table 6-53](#) shows the PIT memory map.

Table 6-53. PIT Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
Periodic Interval Timer (PIT)—Block Base Address 0x0_0400				
0x000	Periodic interval timer control register (PTCNR)	R/W	0x0000_0000	6.6.5.1/6-56
0x004	Periodic interval timer load register (PTLDR)	R/W	0x0000_0000	6.6.5.2/6-56
0x008	Periodic interval timer prescale register (PTPSR)	R/W	0x0000_0000	6.6.5.3/6-57
0x00C	Periodic interval timer counter register (PTCTR)	R	0x0000_0000	6.6.5.4/6-57
0x010	Periodic interval timer event register (PTEVR)	w1c	0x0000_0000	6.6.5.5/6-58
0x014–0x01F	Reserved	—	—	

6.6.5.1 Periodic Interval Timer Control Register (PTCNR)

The periodic interval timer control register (PTCNR), shown in [Figure 6-40](#), is used to enable the different PIT functions. The register can be read at any time.

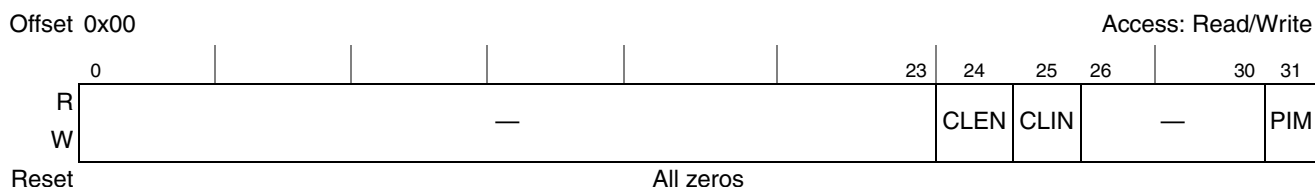


Figure 6-40. Periodic Interval Timer Control Register (PTCNR)

[Table 6-54](#) defines the bit fields of PTCNR.

Table 6-54. 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 RTC_PIT_CLOCK.
26–30	—	Write reserved, read = 0
31	PIM	Periodic interrupt mask bit. Used to enable or disable (mask) the PIT periodic interrupt. 0 Periodic interrupt generation disabled. 1 Periodic interrupt generation enabled.

6.6.5.2 Periodic Interval Timer Load Register (PTLDR)

The periodic interval timer load register (PTLDR), shown in [Figure 6-41](#), contains the 32-bit value to be loaded in a 32-bit PIT counter.

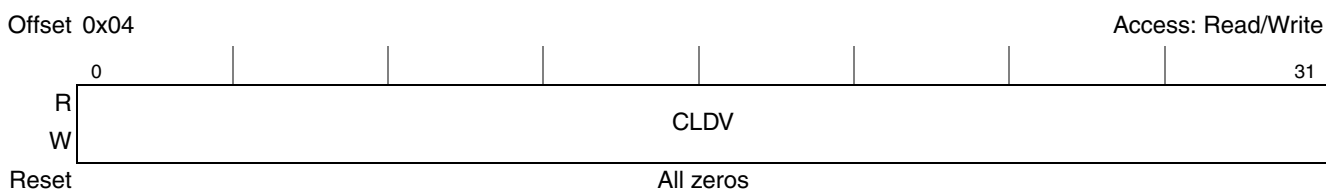


Figure 6-41. Periodic Interval Timer Load Register (PTLDR)

Table 6-55 defines the bit fields of PTLDR.

Table 6-55. PTLDR Bit Settings

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

6.6.5.3 Periodic Interval Timer Prescale Register (PTPSR)

The periodic interval timer prescale register (PTPSR), shown in Figure 6-42, is a read/write register that used to configure the PIT prescaler's value.

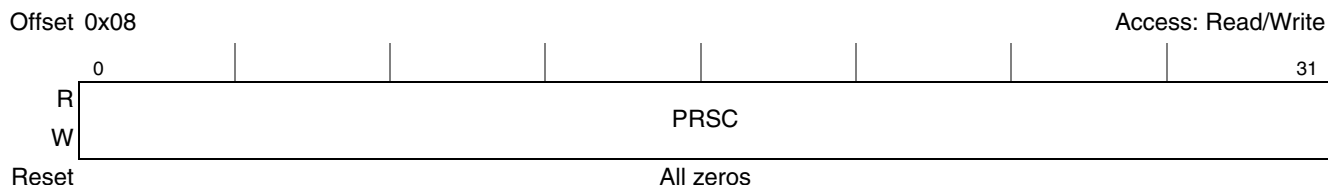


Figure 6-42. Periodic Interval Timer Prescale Register (PTPSR)

Table 6-56 defines the bit fields of PTPSR.

Table 6-56. 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.

6.6.5.4 Periodic Interval Timer Counter Register (PTCTR)

The periodic interval timer counter register (PTCTR), shown in Figure 6-43, is a read-only register that shows the current value in the PIT counter. The PTCTR counter is not affected by reads or writes.

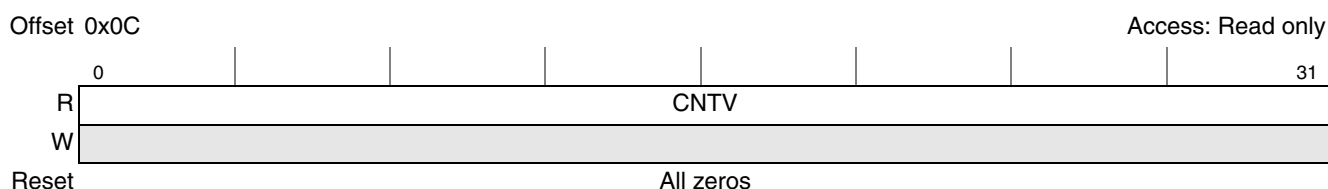


Figure 6-43. Periodic Interval Timer Counter Register (PTCTR)

Table 6-57 defines the bit fields of PTCTR.

Table 6-57. 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].

6.6.5.5 Periodic Interval Timer Event Register (PTEVR)

The periodic interval timer event register (PTEVR), shown in Figure 6-44, 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.

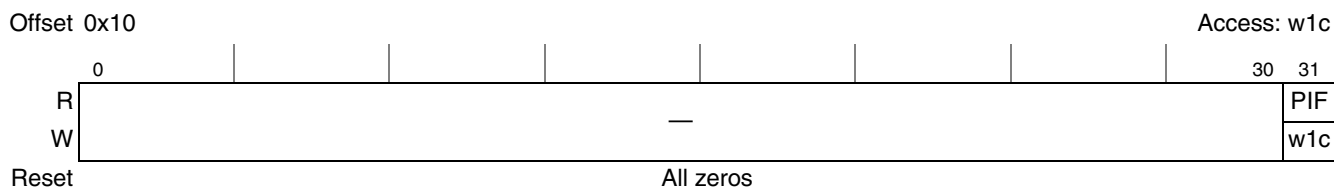


Figure 6-44. Periodic Interval Timer Event Register (PTEVR)

Table 6-58 defines the bit fields of PTEVR.

Table 6-58. PTEVR Bit Settings

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

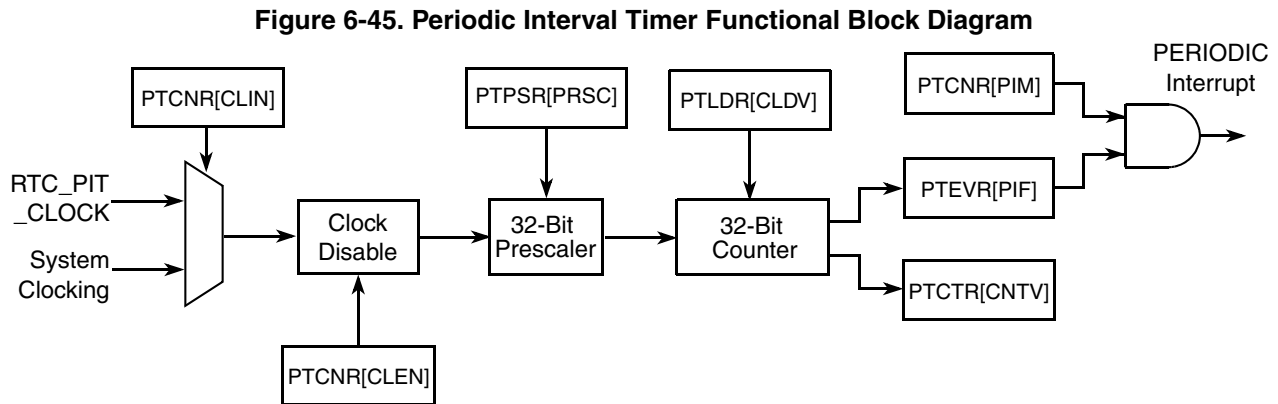
6.6.6 Functional Description

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

6.6.6.1 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 RTC_PIT_CLOCK. The 32-bit counter decrements to zero when loaded with a initial value from the periodic interval timer load register (PTLDR). After the timer reaches zero, PTEVR[PIF] is set and an interrupt is generated if PTCNR[PIM] = 1. At the next count cycle, the value in the PTLDR[CLDV] is loaded into the counter and the process repeats. When a new value is loaded into the PTLDR[CLDV], the PIT is updated, the prescaler counter is reset, and the counter begins counting. Setting of PTEVR[PIF] generates an interrupt, that remains pending until PTEVR[PIF] is cleared. If PTEVR[PIF] is set again before being cleared, the interrupt remains pending until PTEVR[PIF] is cleared. Any write to the PTLDR[CLDV] stops the current countdown and the count resumes with the new value in PTLDR[CLDV]. If PTCNR[CLEN] = 0, the PIT cannot count and retains the old count value. PTCTR contain the PIT current value. The PIT function can be disabled if needed.

Figure 6-45 shows the functional PIT block diagram.



6.6.6.2 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 RTC_PIT_CLOCK.

 - PIT use the internal input clock mode (PTCNR[CLIN] = 0)
 - PIT use the RTC_PIT_CLOCK (PTCNR[CLIN] = 1)

6.6.7 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.

6.7 General-Purpose Timers (GTM)

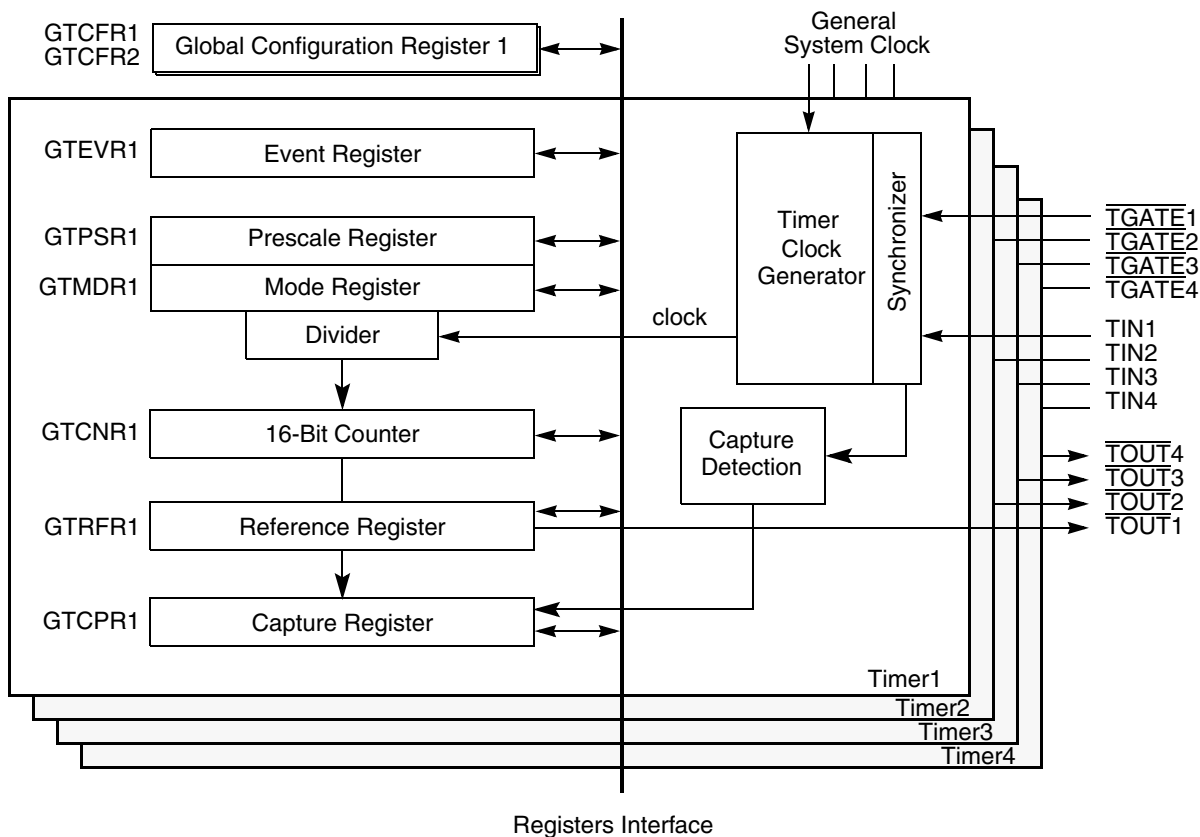
The following sections describe theory of operation of the general purpose (global) timer module, including a definition of the external signals and the functionality. Additionally, the configuration, control, and status registers are described. Note that individual chapters in this book describe additional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

6.7.1 GTM Overview

Each 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 6-46 shows the functional GTM block diagram.

Figure 6-46. Global Timers Block Diagram



6.7.2 GTM Features

The key features of the timer include the following:

- The maximum input clock is the system bus clock
- Four 16-bit programmable timers

- Two timers cascaded internally or externally (using TIN and TOUT external signals) to form a 32-bit timer
- One timer cascaded internally or externally to form a 64-bit timer
- Maximum period of ~549.6 (at 125-MHz bus clock and prescaler = 256) for 16-bit timer
- Maximum period of ~8796 seconds (at 125-MHz bus clock and prescaler = 256) for 32-bit timer
- Maximum period of thousands of years (at 125-MHz bus clock and prescaler = 256) for 64-bit timer
- 8-nanosecond timer resolution (at 125-MHz bus clock and no prescaler)
- Resolution and maximum period can be traded off by selecting prescaler divisor
- Three programmable input clock sources for the timer prescalers
- Input capture capability
- Output compare with programmable mode for the output pin
- Free run and restart modes
- Functional and programming compatibility with MPC8260 timers

6.7.3 GTM Modes of Operation

The GTM unit can operate in the following modes:

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

6.7.3.1 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.

6.7.3.2 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

6.7.3.3 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.

6.7.3.4 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.

6.7.4 GTM External Signal Description

This section provides an overview and detailed descriptions of the GTM signals.

There are four distinct external input timer capture signals (TIN1, TIN2, TIN3, and TIN4), four distinct external input timer gate signals (TGATE1, TGATE2, TGATE3, and TGATE4), and four distinct external timer output signals ($\overline{\text{TOUT1}}$, $\overline{\text{TOUT2}}$, $\overline{\text{TOUT3}}$, and $\overline{\text{TOUT4}}$). The GTM interface signals are defined in [Table 6-59](#).

Table 6-59. GTM Signal Properties

Name	Port	Function	I/O	Reset	Require Pull Up
TIN1	TIN1	Global timer 1 capture control signal	I	0	No
TIN2	TIN2	Global timer 2 capture control signal	I	0	No
TIN3	TIN3	Global timer 3 capture control signal	I	0	No
TIN4	TIN4	Global timer 4 capture control signal	I	0	No

Table 6-59. GTM Signal Properties (continued)

Name	Port	Function	I/O	Reset	Require Pull Up
$\overline{\text{TGATE1}}$	$\overline{\text{TGATE1}}$	Global timer 1 counter gate control signal	I	0	No
$\overline{\text{TGATE2}}$	$\overline{\text{TGATE2}}$	Global timer 2 counter gate control signal	I	0	No
$\overline{\text{TGATE3}}$	$\overline{\text{TGATE3}}$	Global timer 3 counter gate control signal	I	0	No
$\overline{\text{TGATE4}}$	$\overline{\text{TGATE4}}$	Global timer 4 counter gate control signal	I	0	No
$\overline{\text{TOUT1}}$	$\overline{\text{TOUT1}}$	Global timer 1 counter output signal	O	1	No
$\overline{\text{TOUT2}}$	$\overline{\text{TOUT2}}$	Global timer 2 counter output signal	O	1	No
$\overline{\text{TOUT3}}$	$\overline{\text{TOUT3}}$	Global timer 3 counter output signal	O	1	No
$\overline{\text{TOUT4}}$	$\overline{\text{TOUT4}}$	Global timer 4 counter output signal	O	1	No

Table 6-60 provides detailed descriptions of the external GTM signals.

Table 6-60. 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 $\text{GTMDR}_n[\text{CE}]$. 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 $\text{GTCFR}[\text{GM}_x]$ bits. In a restart gate mode ($\text{GTCFR}[\text{GM}_n] = 0$), the $\overline{\text{TGATE}}_n$ pin is used to enable/disable count. A falling $\overline{\text{TGATE}}_n$ pin enables and restarts the count and a rising edge of $\overline{\text{TGATE}}_n$ disables the count. In a normal gate mode ($\text{GTCFR}[\text{GM}_n] = 1$), the $\overline{\text{TGATE}}_n$ have similar functionality, except the falling edge of $\overline{\text{TGATE}}_n$ does not restart the appropriate count value in $\text{GTCNR}_n[\text{CNV}_n]$.
		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 or stops counting depending on the signal state and the configured mode .

Table 6-60. 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 $\text{GTMDR}_n[\text{OM}_n]$. 1. Active-low pulse on $\overline{\text{TOUT}}_n$ for one timer input clock cycle as defined by the $\text{GTMDR}_n[\text{ICLK}_n]$ bits ($\text{GTMDR}_n[\text{OM}_n] = 1$). 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 ($\text{GTMDR}_n[\text{OM}_n] = 0$). $\overline{\text{TOUT}}_n$ changes occur on the rising edge of the timer input clock.
		Timing Assertion/Negation— $\overline{\text{TOUT}}_n$ changes occur on the rising edge of the timer input clock.

6.7.5 GTM Memory Map/Register Definition

The GTM programmable register map occupies 64 bytes of memory-mapped space. Reading undefined portions of the memory map returns all zeros; writing has no effect.

All GTM registers are 8 or 16 bits wide, located on 8-bit or 16-bit address boundaries, and should only be accessed as 8-bit or 16-bit quantities. All addresses used in this chapter are offsets from GTM, as defined in Chapter 2, “Memory Map.”

Table 6-61 shows the memory map of the GTM.

Table 6-61. GTM Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
General Purpose (Global) Timer Module 1—Block Base Address 0x0_0500 General Purpose (Global) Timer Module 1—Block Base Address 0x0_0600				
0x000	Timer 1 and 2 global timers configuration register (GTCFR1)	R/W	0x00	6.7.5.1/6-65
0x001–0x003	Reserved	—	—	—
0x004	Timer 3 and 4 global timers configuration register (GTCFR2)	R/W	0x00	6.7.5.1/6-65
0x005–0x00F	Reserved	—	—	—
0x010	Timer 1 global timers mode register (GTMDR1)	R/W	0x0000	6.7.5.2/6-69
0x012	Timer 2 global timers mode register (GTMDR2)			
0x014	Timer 1 global timers reference register (GTRFR1)	R/W	0xFFFF	6.7.5.3/6-70
0x016	Timer 2 global timers reference register (GTRFR2)			
0x018	Timer 1 global timers capture register (GTCPR1)	R/W	0x0000	6.7.5.4/6-70
0x01A	Timer 2 global timers capture register (GTCPR2)			
0x01C	Timer 1 global timers counter register (GTCNR1)	R/W	0x0000	6.7.5.5/6-71
0x01E	Timer 2 global timers counter register (GTCNR2)			

Table 6-61. GTM Register Address Map (continued)

Offset	Register	Access	Reset Value	Section/ Page
0x020	Timer 3 global timers mode register (GTMDR3)	R/W	0x0000	6.7.5.2/6-69
0x022	Timer 4 global timers mode register (GTMDR4)			
0x024	Timer 3 global timers reference register (GTRFR3)	R/W	0xFFFF	6.7.5.3/6-70
0x026	Timer 4 global timers reference register (GTRFR4)			
0x028	Timer 3 global timers capture register (GTCPR3)	R	0x0000	6.7.5.4/6-70
0x02A	Timer 4 global timers capture register (GTCPR4)			
0x02C	Timer 3 global timers counter register (GTCNR3)	R/W	0x0000	6.7.5.5/6-71
0x02E	Timer 4 global timers counter register (GTCNR4)			
0x030	Timer 1 global timers event register (GTEVR1)	w1c	0x0000	6.7.5.6/6-71
0x032	Timer 2 global timers event register (GTEVR2)			
0x034	Timer 3 global timers event register (GTEVR3)			
0x036	Timer 4 global timers event register (GTEVR4)			
0x038	Timer 1 global timers prescale register (GTPSR1)	R/W	0x0003	6.7.5.7/6-72
0x03A	Timer 2 global timers prescale register (GTPSR2)			
0x03C	Timer 3 global timers prescale register (GTPSR3)			
0x03E	Timer 4 global timers prescale register (GTPSR4)			
General Purpose (Global) Timer Module 2: All registers defined for GTM1 are also defined for GTM2; the base address of GTM2 registers is 0x0_0600. The offset for the registers of Global Timer Module 2 timers (GTM 5-8) remains the same.				

6.7.5.1 Global Timers Configuration Registers (GTCFR_n)

The global timers configuration registers (GTCFR1 and GTCFR2), shown in [Figure 6-47](#) and [Figure 6-48](#), 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

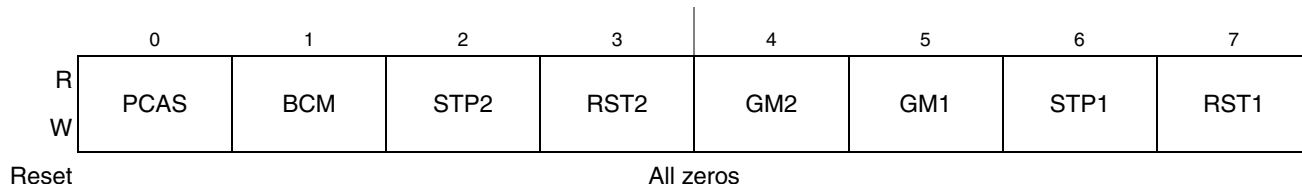


Figure 6-47. Global Timers Configuration Register 1 (GTCFR1)

Table 6-62 defines the bit fields of GTCFR1.

Table 6-62. 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 controls 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 $\overline{TGATE2}$ 0 Restart gate mode. The $\overline{TGATE2}$ pin is used to enable/disable count. A low level of $\overline{TGATE2}$ enables and a falling edge of $\overline{TGATE2}$ restarts the count (reset the dynamic counter's count value to 0) and a high level of $\overline{TGATE2}$ disables the count. 1 Normal gate mode. This mode is the same as 0, except the falling edge of $\overline{TGATE2}$ does not restart the appropriate count value in $GTCNR2[CNV2]$.

Table 6-62. GTCFR1 Bit Settings (continued)

Bits	Name	Description
5	GM1	<p>Gate mode for $\overline{\text{TGATE1}}$</p> <p>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.</p> <p>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].</p> <p>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.</p>
6	STP1	<p>Stop timer 1</p> <p>0 Normal operation</p> <p>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.</p>
7	RST1	<p>Reset timer 1</p> <p>0 Reset the timer 1, including GTMDR1, GTRFR1, GTCNR1, GTCPR1, and GTEVR1 (a software reset is identical to an external reset).</p> <p>1 Enable the corresponding timer if the STP1 bit is cleared.</p>

The GTCFR2 register is shown in [Figure 6-48](#).

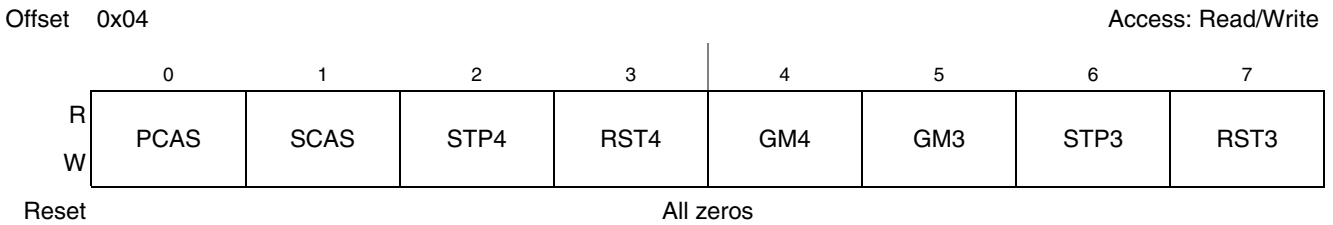


Figure 6-48. Global Timers Configuration Register 2 (GTCFR2)

Table 6-63 defines the bit fields of GTCFR2.

Table 6-63. GTCFR2 Bit Settings

Bits	Name	Description
0	PCAS	<p>Pair-cascade mode</p> <p>0 Normal operation.</p> <p>1 Timers 3 and 4 cascade to form a 32-bit timer.</p> <p>Note: This bit is ignored in super-cascade mode (GTCFR2[SCAS] = 1).</p> <p>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.</p>
1	SCAS	<p>Super cascade mode</p> <p>0 Normal operation</p> <p>1 Timers 1, 2, 3 and 4 cascade to form a 64-bit timer.</p> <p>Note: In super-cascade mode (GTCFR2[SCAS] = 1) the pair-cascade mode bits are ignored, (GTCFR1/2[PCAS] = Don't Care).</p> <p>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.</p>
2	STP4	<p>Stop timer 4</p> <p>0 Normal operation</p> <p>1 Reduce power consumption of the corresponding timer. This bit stops all clocks to the timer 4, except the register interface clock, which allows to read and write timer registers. The clocks to the timer remain stopped until the user clears this bit or a hardware reset occurs.</p>
3	RST4	<p>Reset timer 4</p> <p>0 Reset the timer 4, including GTMDR4, GTRFR4, GTCNR4, GTCPR4, and GTEVR4 (a software reset is identical to an external reset).</p> <p>1 Enable the corresponding timer if the STP4 bit is cleared.</p>
4	GM4	<p>Gate mode for $\overline{\text{TGATE4}}$</p> <p>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.</p> <p>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].</p>
5	GM3	<p>Gate mode for $\overline{\text{TGATE3}}$</p> <p>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.</p> <p>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].</p> <p>Note: In backward compatible mode (GTCFR1[BCM] = 0) this bit is ignored. The GTCFR2[GM4] bit controls the gate mode for timers 3 and 4.</p>
6	STP3	<p>Stop timer 3</p> <p>0 Normal operation</p> <p>1 Reduce power consumption of the corresponding timer. This bit stops all clocks to the timer 3, except the register interface clock, which allows to read and write timer registers. The clocks to the timer remain stopped until the user clears this bit or a hardware reset occurs.</p>
7	RST3	<p>Reset timer 3</p> <p>0 Reset the timer 3, including GTMDR3, GTRFR3, GTCNR3, GTCPR3, and GTEVR3 (a software reset is identical to an external reset).</p> <p>1 Enable the corresponding timer if the STP3 bit is cleared.</p>

6.7.5.2 Global Timers Mode Registers (GTMDR1–GTMDR4)

The global timers mode registers (GTMDR1, GTMDR2, GTMDR3, and GTMDR4) are shown in Figure 6-49.

Erratic behavior may occur if GTCFR1 and GTCFR2 are not initialized before the GTMDR n . Only GTCFR n [RST n] and GTCFR n [STP n] can be modified at any time.

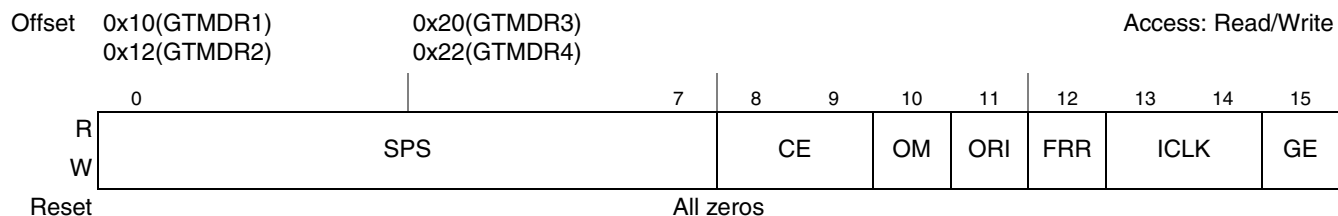


Figure 6-49. Global Timers Mode Registers (GTMDR1–GTMDR4)

Table 6-64 defines the bit fields of GTMDR.

Table 6-64. 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 on reaching the reference value.
12	FRR	Free run/restart mode 0 Free run. The timer count continues to increment after the reference value is reached. 1 Restart. The timer count is reset immediately after the reference value is reached.

Table 6-64. GTMDR Bit Settings (continued)

Bits	Name	Description
13–14	ICLK	Input clock source for the timer. 00 Internally cascaded input. This selection means: For ICLK1, the timer 1 input is the output of timer 2. For ICLK2, the timer 1 input is the output of timer 2, the timer 2 input is the output of timer 3, the timer 3 input is the output of timer 4. For ICLK3, the timer 3 input is the output of timer 4. For ICLK4 this selection means no input clock is provided to the timer. 01 Internal general system bus clock. 10 Internal slow go clock (divided by 16 system bus clock). 11 TIN n : corresponding TIN1, TIN2, TIN3, or TIN4 pin (falling edge).
15	GE	Gate enable 0 The $\overline{\text{TGATE}}_n$ signal is ignored. 1 The $\overline{\text{TGATE}}_n$ signal is used to control the timer.

6.7.5.3 Global Timers Reference Registers (GTRFR1–GTRFR4)

Global timers reference registers, shown in [Figure 6-50](#), are 16-bit memory-mapped, read/write registers containing the 16-bit reference values for each timer’s timeout. The reference value is not reached until $\text{GTCNR}_n[\text{CNV}]$ increments to the value in $\text{GTRFR}_n[\text{TRV}]$.

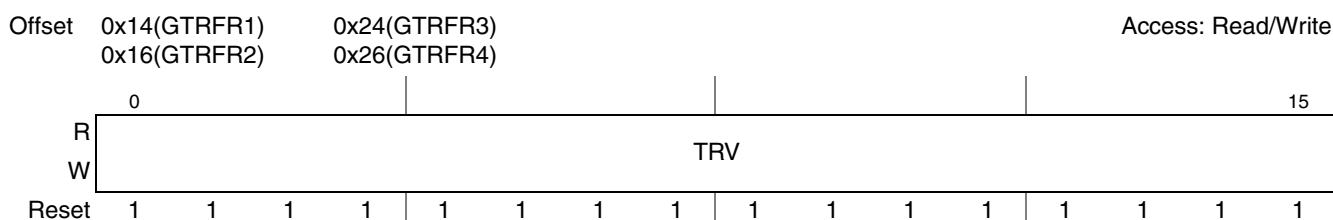


Figure 6-50. Global Timers Reference Registers (GTRFR1–GTRFR4)

[Table 6-65](#) defines the bit fields of GTRFR.

Table 6-65. 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.

6.7.5.4 Global Timers Capture Registers (GTCPR1–GTCPR4)

Global timers capture registers (GTCPR_1 , GTCPR_2 , GTCPR_3 , and GTCPR_4), shown in [Figure 6-51](#), are used to latch the value of the counters according to $\text{GTMDR}_n[\text{CE}]$.

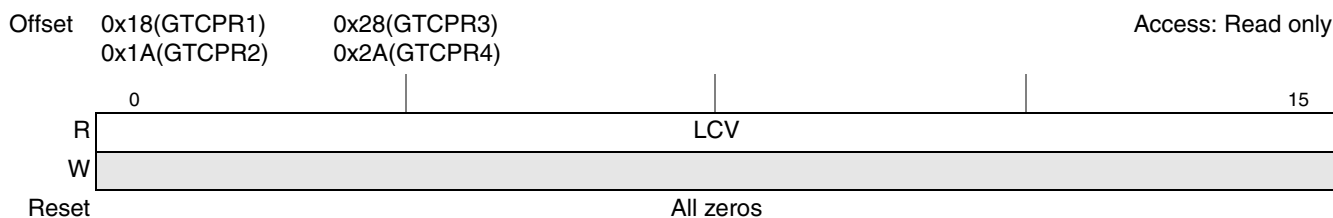


Figure 6-51. Global Timers Capture Registers (GTCPR1–GTCPR4)

Table 6-66 defines the bit fields of $GTCPR_n$.

Table 6-66. $GTCPR_n$ Bit Settings

Bits	Name	Description
0–15	LCV	Latched counter value. Corresponding timer's 16-bit latched value.

6.7.5.5 Global Timers Counter Registers ($GTCNR_1$ – $GTCNR_4$)

Global timers counter registers ($GTCNR_1$, $GTCNR_2$, $GTCNR_3$, and $GTCNR_4$), shown in Figure 6-52, 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.

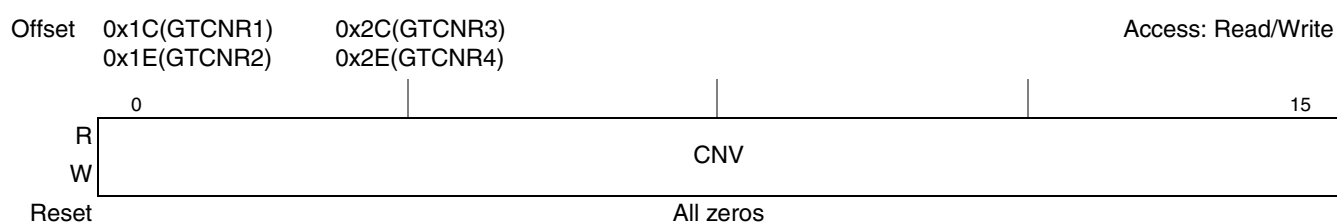


Figure 6-52. Global Timers Counter Registers ($GTCNR_1$ – $GTCNR_4$)

Table 6-67 defines the bit fields of $GTCNR$.

Table 6-67. $GTCNR$ Bit Settings

Bits	Name	Description
0–15	CNV	Counter value. Corresponding timer's 16-bit read/write up-counter value.

6.7.5.6 Global Timers Event Registers ($GTEVR_1$ – $GTEVR_4$)

Global timers event registers ($GTEVR_1$, $GTEVR_2$, $GTEVR_3$, and $GTEVR_4$), shown in Figure 6-53, 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 $GTMDR_n[CE]$. $GTEVR$ s 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.

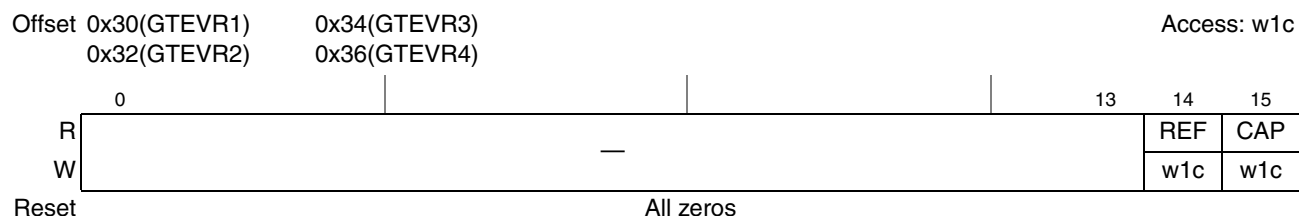


Figure 6-53. Global Timers Event Registers ($GTEVR_1$ – $GTEVR_4$)

Table 6-68 defines the bit fields of GTEVR_n.

Table 6-68. 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.

6.7.5.7 Global Timers Prescale Registers (GTPSR1–GTPSR4)

The global timers prescale registers (GTPSR1, GTPSR2, GTPSR3, and GTPSR4) are shown in Figure 6-54.

Erratic behavior may occur if GTPSR_n is not initialized before the corresponding GTMDR_n.

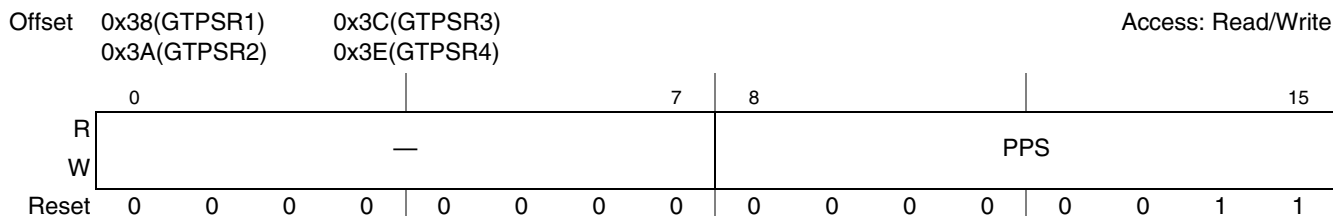


Figure 6-54. Global Timers Prescale Registers (GTPSR1–GTPSR4)

Table 6-69 defines the bit fields of GTPSR_n.

Table 6-69. GTPSR_n Bit Settings

Bits	Name	Description
0–7	—	Reserved, should be cleared.
8–15	PPS	Primary prescaler bits The primary prescaler is programmed to divide the clock input to corresponding timer by values from 1 to 256. The value 0x00 divides the clock by 1 and 0xFF divides the clock by 256.

NOTE

The total timer prescale value is calculated as follows:

$$GTM_{n_prescaler} = (GTPSR_n[PPS] + 1) \cdot (GTMDR_n[SPS] + 1)$$

This gives a total prescale range from 1 (GTPSR_n[PPS] = 0x00, GTMDR_n[SPS] = 0x00) to 65,536 (GTPSR_n[PPS] = 0xFF, GTMDR_n[SPS] = 0xFF).

6.7.6 Functional Description

This section provides a functional description of the general-purpose timer units, including detailed description of its modes of operation.

6.7.6.1 General-Purpose Timer Units

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

- The system clock
- The system slow go clock (internally divided by 16)

The general system clock is generated in the clock synthesizer and defaults to the system frequency. However, the general system clock has the option to be divided before it leaves the clock synthesizer. This mode, called slow go, is used to save power. Whatever the resulting frequency of the general system clock, the user can either choose that frequency or the frequency divided by 16 as the input to the prescaler of each timer. Alternatively, the user may prefer TIN_n to be the clock source. TIN_n is internally synchronized to the internal clock. If the user has chosen to internally cascade two 16-bit timers to a 32-bit timer, then a timer can use the clock generated by the output of another timer.

The clock input source is selected by the corresponding $GTMDR_n[ICLK]$ bits. The prescalers ($GTMDR_n[SPS]$ and $GTPSR_n[PPS]$) can be programmed to divide the clock input by values from 1 to 65,536 and the output of the prescaler is used as an input to the 16-bit counters. The best resolution of the timer is one clock cycle (8 ns at a 125-MHz system clock, for example). The maximum period (when the reference value is all ones and the prescaler divides by 256×256 , and slow go mode divides by 16) for one 16-bit timer is ~549.6 sec at 125 MHz.

6.7.6.2 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 $GTMR$ selects each mode.

- Free run reference mode ($GTMDR_n[FRR] = 0$)
The corresponding timer count continues to increment after the reference value is reached.
- Reset reference mode ($GTMDR_n[FRR] = 1$)
The corresponding timer count is reset immediately after the reference value is reached.

Upon reaching the reference value, the corresponding $GTEVR_n[REF]$ bit is set and an interrupt is issued if $GTMDR_n[ORI] = 1$. The timers can output a signal on the timer output pin \overline{TOUT}_n if the reference value is reached (selected by the corresponding $GTMDR_n[OM]$). This signal can be an active-low pulse or a toggle of the current output. The output can also be connected internally to the input of another timer, resulting in a 32- or 64-bit timer.

6.7.6.3 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{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 GTCFR n .

NOTE

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

6.7.6.4 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 6-55). When working in the none-cascaded mode, the non-cascaded GTRFR, GTCPR, and GTCNR should be referenced with appropriate 16-bit bus cycles.

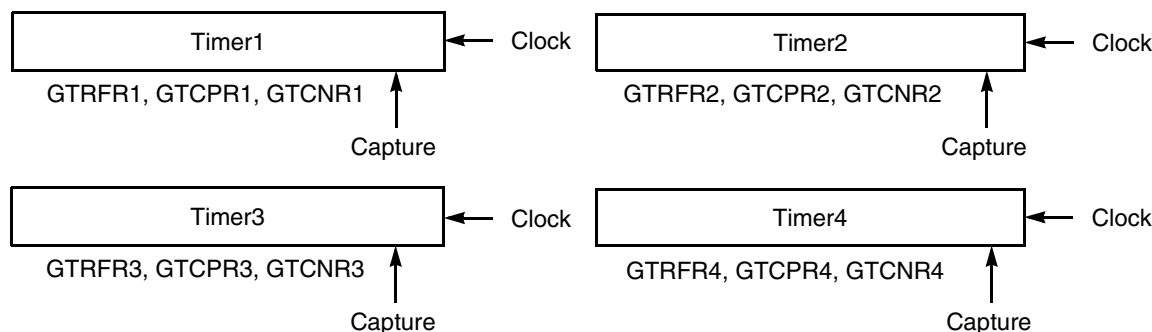


Figure 6-55. Timers Non-Cascaded Mode Block Diagram

- 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 6-56](#). 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 $GTCFR1[PCAS] = 0$, $GTCFR2[PCAS] = 1$), or two 32-bit timers ($GTCFR1[PCAS] = 1$ and $GTCFR2[PCAS] = 1$).

If $GTCFR1[PCAS] = 1$ and/or $GTCFR2[SCAS] = 1$, the two 16-bit timers (timer 1 and timer 2 or timer 3 and timer 4) function as a 32-bit timer with a 32-bit GTRFR, GTCPR, and GTCNR. In this case, GTMDR1/GTMDR3 is ignored, and the modes and functions are defined using GTMDR2/GTMDR4, and GTCFR1/GTCFR2. The capture are controlled from TIN2, and the interrupts are generated from GTEVR2. When working in the pair-cascaded mode, the cascaded GTRFR, GTCPR, and GTCNR should be referenced with 32-bit bus cycles.

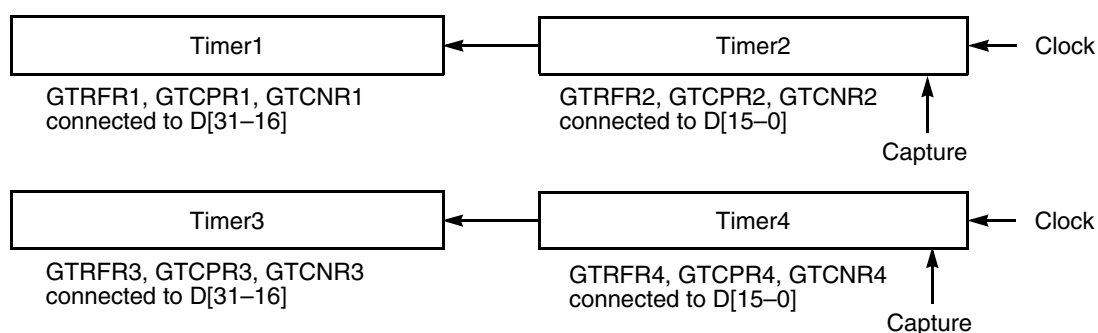


Figure 6-56. Timer Pair-Cascaded Mode Block Diagram

- 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 6-57](#).
 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.

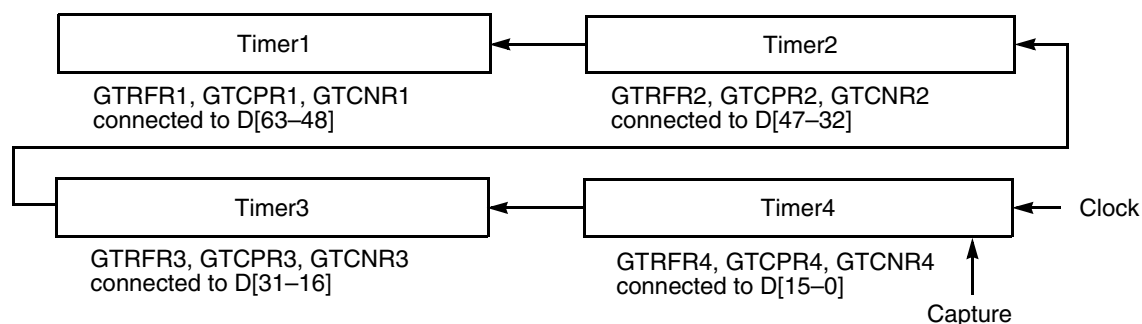


Figure 6-57. Timers Super-Cascaded Mode Block Diagram

6.7.7 Initialization/Application Information (Programming Guidelines for GTM Registers)

The following initialization sequence of GTM is recommended:

- Write to $GTCFR_n$ in order to reset, to stop or to configure the appropriate timer's operation: cascaded timers configuration, gate mode configuration.
- Write to $GTPSR_n[PPS]$ fields in order to program the appropriate timer's clock primary prescaler.
- Write to $GTMDR_n$ in order to choose an input clock, to program the secondary prescaler and to set a desirable appropriate timer's operational mode.

NOTE

Erratic behavior may occur if $GTCFR_n$ and $GTPSR$ are not initialized before the $GTMDR$. Only $GTCFR_n[RST_n]$ can be modified at any time

- Clear $GTEVR_n[REF]$ and $GTEVR_n[CAP]$ by writing 1s in order to clear the previous events.
- Write to $GTRFR$ and to $GTCNR_n$ according to appropriate timer's $GTMDR_n$ programming.

NOTE

A write cycle to a $GTCNR_n[CNV]$ fields sets the register to the written value, causing its corresponding primary and secondary prescalers, ($GTPSR_n[PPS]$ and $GTMDR_n[SPS]$), to be reset.

- Write to $GTCFR_n[STP_n]$ and to $GTCFR_n[RST_n]$ in order to initialize the appropriate timer's operation.
- Write to $GTCFR_n$ register in order to start the corresponding timer ($GTCFR_n[STP_n] = 0$).

6.8 Power Management Control (PMC)

The device provides a power management control (PMC) unit, which enables the device to smoothly enter and exit low power modes. Low power modes may be used when internal units in the device temporarily or permanently do not perform any action.

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

- Dynamic power management
- Shutting down clocks to unused blocks
- Software-controlled power-down states for the e300 core

6.8.1 External Signal Description

Table 6-70 describes the power management signals.

Table 6-70. System Control Signals—Detailed Signal Descriptions

Signal	I/O	Description
$\overline{\text{QUIESCE}}$	O	Quiesce state. Indicates that the processor system and e300 core are in low power state.
		State Meaning Asserted—The system and e300 core are in low power state. Negated—The system and e300 core are not in low power state.
		Timing The timing between a quiesce request from the e300 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.

6.8.2 PMC Memory Map/Register Definition

Table 6-71 shows the memory map for the power management controller registers.

Table 6-71. Power Management Controller Registers Memory Map

Offset	Register	Access	Reset	Section/Page
PMC—Block Base Address 0x0_0B00				
0x00B00	Power management controller configuration register (PMCCR)	R/W	0x0000_0000	6.8.2.1/6-77
0x00B04–0x00BFC	Reserved	—	—	—

6.8.2.1 Power Management Controller Configuration Register (PMCCR)

The power management controller configuration register (PMCCR), shown in Figure 6-58, controls whether only the e300 core will enter low power state upon quiesce request or additional parts of the device will also enter low power state.

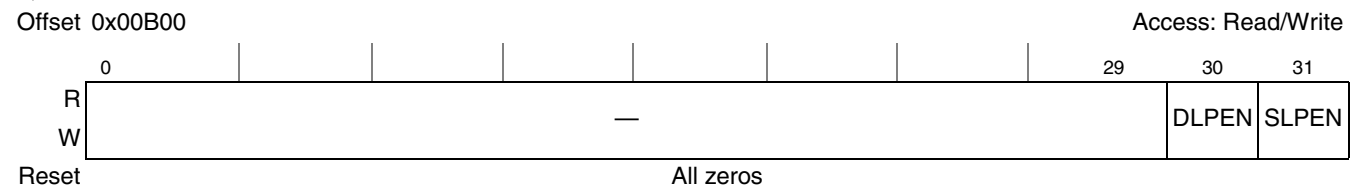


Figure 6-58. Power Management Controller Configuration Register

Table 6-5 defines the bit fields of PMCCR.

Table 6-72. 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 signal is generated from e300 core. This bit is cleared when the device exits from low power state.

6.8.3 Functional Description

The device has features to minimize power consumption at several levels. Software can shut down clocks to individual blocks when they are not needed through a memory-mapped register in the clock unit (SCCR). Additionally, software running on the e300core can access the core’s SPRs to put the device into doze, nap, or sleep power down states. These power management features are described in further detail in this section.

There are four power states in the MPC8309:

- D0 : Full-power state
- D1 : Core in doze mode, e300 PLL running
- D2 : Core in nap mode, e300 PLL running
- D3: Core in sleep mode, e300 PLL not running

Table 6-73. Software-Controller Power-Down States—Basic Description

System Mode	Core Mode	Description	Core Responds To		DDR SDRAM State	Quiesce Signal State
			Snoop	Interrupt		
Full On (PMCCR[SLPEN] = 0)	Full On	The e300 core is operating normally	Yes	Yes	Active	Negated
	Doze	Core stops dispatching new instructions (core is halted), and most of the core functional units are disabled. System operates normally.	Yes	Yes	Active	Negated
	Nap	Core is stopped with its clocks off except to time base. System operates normally.	No	Yes	Active	Negated
	Sleep	Core is stopped with its clocks off. Core clocks to all blocks (including core time base) are stopped except to the interrupt unit. System operates normally.				

Table 6-73. Software-Controller Power-Down States—Basic Description (continued)

System Mode	Core Mode	Description	Core Responds To		DDR SDRAM State	Quiesce Signal State
			Snoop	Interrupt		
Low Power (PMCCR[SLPEN] = 1)	Nap	Core operation as described above. System is in idle state, DDR SDRAM memory operates in self-refresh mode if enabled.	No	Yes	According to PMCCR [DLPEN]	Asserted
	Sleep					

6.8.3.1 Shutting Down Clocks to Unused Blocks

As described in [Section 4.5.2.3, “System Clock Control Register \(SCCR\),”](#) SCCR provides a way to shut down certain functional blocks within the device when they are not needed in a particular system. SCCR can be written by the e300 core or by an external master. Powering down a block in this way turns off all clocks to that block. It does not remove power. It is required that the SCCR is written to shut down a certain functional block only when that block is idle.

NOTE

Functional blocks disabled using SCCR cannot respond to configuration accesses. Any access to configuration, control, and status registers of a disabled block is a programming error.

6.8.3.2 Software-Controlled Power-Down States

e300 core software can place the core in doze, nap, or sleep power-down states by writing to `HID0` in the core, as described in detail in the section “Hardware Implementation Register 0 (HID0),” of the *e300 PowerPC Core Reference Manual*. In addition, if `PMCCR[SLPEN]` is set when the e300 core request to enter nap or sleep modes, it also causes the DDR2 controller to enter low power mode. The basic relationships between core and system power management states is shown in [Table 6-73](#).

To preserve cache coherency and otherwise avoid loss of system state, the core transitions to low-power modes is coordinated with DDR2 controller. The power management controller allows the core to enter power down mode only when the rest of the system is idle.

When the power management controller detects that the internal system bus is idle, and there are no outstanding transactions, it signals the internal logic units to enter low power state.

If `PMCCR[DLPEN]` and/or `PMCCR[SLPEN]` is set, the DDR SDRAM and/or secondary DDR SDRAM is first set to self-refresh mode (if enabled by `DDR_SDRAM_CFG[SREN]` memory controller register) before the memory controller stops driving refresh commands. Self-refresh mode guarantees that the memory content remains valid while the memory controller and its clocks are off. The DDR clocks are then disabled. The DLL is then shut off and stops driving clocks on `MCKn` pins. Finally the DDR SDRAM memory controller enters low power state and acknowledges the power management controller.

The power management controller then signals the core and acknowledges it’s request to enter power down mode. Finally the $\overline{\text{QUIESCE}}$ output signal is asserted.

6.8.3.3 Exiting Core and System Low Power States

The device can exit low power state and return to full-on mode for one of the following reasons:

- The core internal time base unit invokes a request to exit low power state.
- The power management controller has detected that the system is not idle and there are outstanding transactions on the internal bus.

The actions taken to exit low power state depend on the mode and whether the system or only the core are in this state. The following sections describe the various scenarios.

6.8.3.3.1 Exiting Low Power States—Core-Only Mode

Exit from Doze mode is controlled only by the core itself and does not involve the power management controller or other blocks in the device.

Nap mode is exited according to the internal time base unit of the core. When the core returns to full-on state, it signals to the power management controller that it is ready and is immediately acknowledged to access the DDR2 controller.

6.8.3.3.2 Exiting Low Power States—Core and System Mode

The power management controller decides to exit low power state when it detects that the system is not idle anymore. The device may exit idle state when one of the peripheral interfaces makes a request to access the internal bus or when the core returns to full-on state, as described above, and makes a request to access the internal bus. For example, the QUICC Engine block receives an Ethernet frame, and requires to store it on the DDR SDRAM memory.

If the particular DDR SDRAM memory controller is in low power state (PMCCR[DLPEN] was set when entering low power state), the power management controller initially enables the DDR SDRAM memory controller and its DLL, allowing it to lock. When locked, DDR SDRAM clocks (MCK n) are enabled and the memory controller exit self-refresh and returns to auto-refresh mode.

The power management controller then enables other internal units and interrupts the e300 core. When all internal units, including the 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.

Chapter 7

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.

7.1 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)

7.1.1 Coherent System Bus Overview

The coherent system bus is the central bus of the device. Any data transaction from master to slave in the device passes through the coherent system bus. The device coherent system bus supports pipelined transactions. It has independent address and data tenures. Pipeline depth determines the number of address tenures that can be started before the first data tenure is finished.

Basic burst size is equal to cache line length of the core, which is 32 bytes. Using repeat request mode enables up to eight consecutive bursts to be executed by the same master. The maximum number of

consecutive transactions can be limited by programming arbiter configuration register. See [Section 7.2.1, “Arbiter Configuration Register \(ACR\),”](#) for more details.

NOTE

Write accesses to different interfaces are not guaranteed to finish in order.

7.2 Arbiter Memory Map/Register Definition

[Table 7-1](#) shows the memory map for arbiter’s configuration, control and status registers.

Table 7-1. Arbiter Register Map

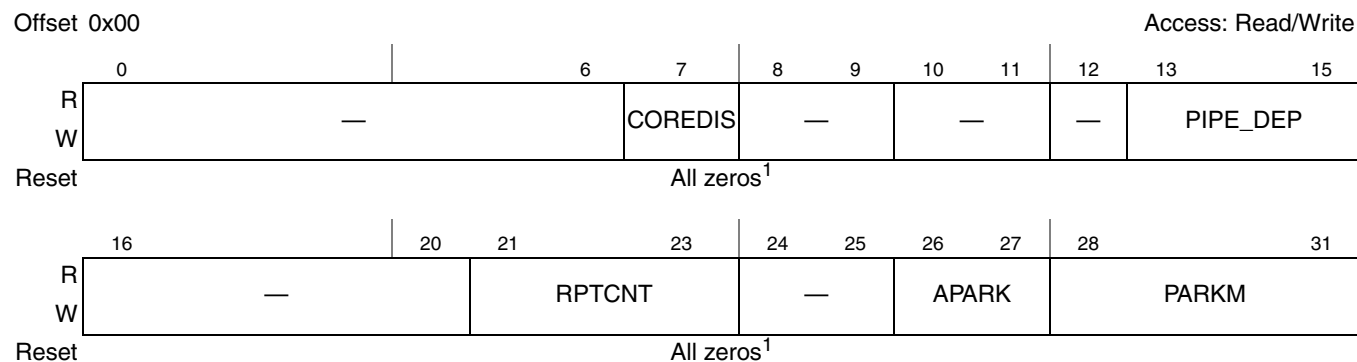
System Arbiter—Block Base Address 0x0_0800				
Memory Offset (Hex)	Register	Access	Reset	Section/Page
0x00	Arbiter configuration register (ACR)	R/W	0x0000_0000/ 0x0010_0000 ¹	7.2.1/7-3
0x04	Arbiter timers register (ATR)	R/W	FFFF_FFFF	7.2.2/7-4
0x0C	Arbiter event register (AER)	w1c	0x0000_0000	7.2.3/7-5
0x10	Arbiter interrupt definition register (AIDR)	R/W	0x0000_0000	7.2.4/7-6
0x14	Arbiter mask register (AMR)	R/W	0x0000_0000	7.2.5/7-7
0x18	Arbiter event attributes register (AEATR)	R	0x0000_0000 ²	7.2.6/7-8
0x1C	Arbiter event address register (AEADR)	R	0x0000_0000 ²	7.2.7/7-9
0x20	Arbiter event response register (AERR)	R/W	0x0000_0000	7.2.8/7-10

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 $\overline{\text{PORESET}}$.

7.2.1 Arbiter Configuration Register (ACR)

Arbiter configuration register (ACR) defines the arbiter modes and parked master on the bus. [Figure 7-1](#) shows the fields of ACR.



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 7-1. Arbiter Configuration Register (ACR)

[Table 7-2](#) describes ACR fields.

Table 7-2. ACR Field Descriptions

Bits	Name	Description
0–6	—	Write reserved, read = 0
7	COREDIS	Core disable. Specifies whether CPU is disabled. When CPU is disabled, it cannot be granted on the bus by the arbiter. After reset, this bit receives its value from the reset configuration bit of COREDIS and can be configured by software. Also, if boot source is boot sequencer, COREDIS must be set to 1 at reset and the last transaction of the boot sequencer must set COREDIS to 0, if CPU enable is needed. 0 CPU enabled. 1 CPU disabled.
8–9	—	Write reserved, read = 0
10–11	—	Reserved. Write should preserve reset value. The reset value is a function of the core PLL configuration, which is part of the reset configuration word. When the core is set to operate at 1:1 or 3:2 bus clock, these bits are set to ‘01’ during reset; otherwise, they are set to ‘00’.
12	—	Write reserved, read = 0
13–15	PIPE_DEP	Pipeline depth (number of outstanding transactions). 000 Pipeline depth 1 (1 outstanding transaction) 001 Pipeline depth 2 (2 outstanding transactions) 010 Pipeline depth 3 (3 outstanding transactions) 011 Pipeline depth 4 (4 outstanding transactions) 1xx Reserved
16–20	—	Write reserved, read = 0

Table 7-2. ACR Field Descriptions (continued)

Bits	Name	Description
21–23	RPTCNT	Repeat count. Specifies the maximum number of consecutive transactions, that any master can perform, using $\overline{\text{REPEAT}}$ request mode. 000 1 consecutive transactions ($\overline{\text{REPEAT}}$ request mode disable) 001 2 consecutive transactions 010 3 consecutive transactions 011 4 consecutive transactions 100 5 consecutive transactions 101 6 consecutive transactions 110 7 consecutive transactions 111 8 consecutive transactions Note: It is recommended not to program this field for more than four consecutive transactions.
24–25	—	Write reserved, read = 0
26–27	APARK	Address parking. Specifies arbiter bus parking mode. 00 Park to master. Arbiter parks the address bus to the master, that is selected by numeric value of PARKM field. 01 Park to last owner. Arbiter parks the address bus to last bus owner. 10 Disable. Arbiter does not assert $\overline{\text{BG}}$ to any master, if no $\overline{\text{BR}}$ is present. 11 Reserved
28–31	PARKM	Parking master. 0000 e300 core 0001 PCI/IOS 0010 Reserved 0011 USB/I2C1_BOOT 0100 DMA, eSDHC 0101–0110 Reserved 0111 CE_COMPLEX_WRAPPER 1000–1111 Reserved

7.2.2 Arbiter Timers Register (ATR)

The arbiter timers register (ATR) defines the arbiter address time out (ATO) and data time out (DTO) values. [Figure 7-2](#) shows the fields of ATR.

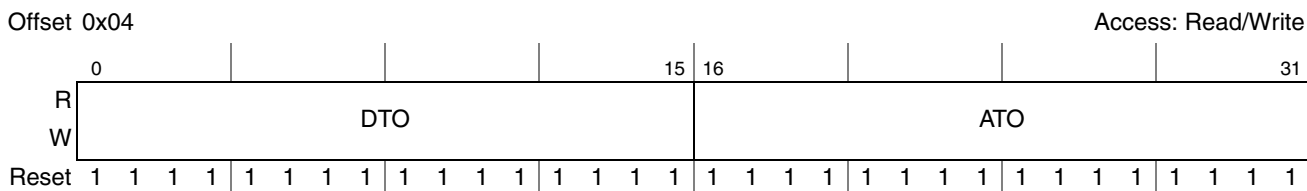


Figure 7-2. Arbiter Timers Register (ATR)

Table 7-3 describes ATR fields.

Table 7-3. ATR Field Descriptions

Bits	Name	Description
0–15	DTO	Data time out. Specifies the time-out period for the data tenure. The granularity of this field is 128 bus clocks. The maximum value is 8355840 coherent system bus clocks. Data time_out occurs if the data tenure does not end before the specified time-out period. When DTO = n, the timeout cycle is n × 128. 0000 Reserved 0001 128 clock cycles 0002 256 clock cycles 0003 384 clock cycles ... FFFF 8355840 clock cycles
16–31	ATO	Address time out. Specifies the time-out period for the address tenure. The granularity of this field is 128 bus clocks. Maximum value is 8355840 coherent system bus clocks. Address time-out occurs if the address tenure did not end before the specified time-out period. When ATO = n, the timeout cycle is n × 128. 0000 Reserved 0001 128 clock cycles 0002 256 clock cycles 0003 384 clock cycles ... FFFF 8355840 clock cycles

7.2.3 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 7-3 shows the fields of AER.

Offset 0x0C

Access: W1c

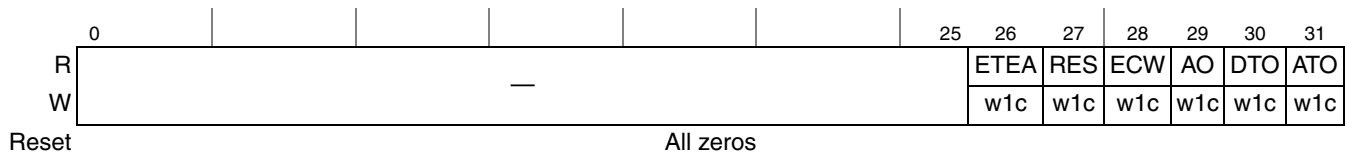


Figure 7-3. Arbiter Event Register (AER)

Table 7-4 describes AER fields.

Table 7-4. AER Field Descriptions

Bits	Name	Description
0–25	—	Write reserved, read = 0
26	EAEA	Transfer error. Reports on detection of transfer error by one of the slaves. 0 No transfer error detected by one of the slaves. 1 Transfer error detected by one of the slaves.
27	RES	Reserved transfer type. Reports on transaction with reserved transfer type. See Section 7.3.2.5, “Reserved Transaction Type,” for more information. 0 No transaction with reserved transfer type occurred. 1 Transaction with reserved transfer type occurred.

Table 7-4. AER Field Descriptions (continued)

Bits	Name	Description
28	ECW	External control word transfer type. Reports on transaction with external control word transfer type. See Section 7.3.2.6, “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 7.3.2.4, “Address Only Transaction Type,” for more information. 0 No transaction with address only transfer type occurred. 1 Transaction with address only transfer type occurred.
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.

7.2.4 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 7-4](#) shows the fields of AIDR.

Offset 0x10

Access: Read/Write



Figure 7-4. Arbiter Interrupt Definition Register (AIDR)

[Table 7-5](#) describes AIDR fields.

Table 7-5. AIDR Field Descriptions

Bits	Name	Description
0–25	—	Write reserved, read = 0
26	ETEA	Transfer error. Detection of transfer error by one of the slaves interrupt definition. 0 Detection of transfer error by one of the slaves causes regular interrupt. 1 Detection of transfer error by one of the slaves causes MCP interrupt.
27	RES	Reserved transfer type. Transaction with reserved transfer type interrupt definition. 0 Transaction with reserved transfer type causes regular interrupt. 1 Transaction with reserved transfer type causes MCP interrupt.
28	ECW	External control word transfer type. Transaction with external control word transfer type interrupt definition. 0 Transaction with external control word transfer type causes regular interrupt. 1 Transaction with external control word transfer type causes MCP interrupt.

Table 7-5. AIDR Field Descriptions (continued)

Bits	Name	Description
29	AO	Address only transfer type. Transaction with address only transfer type interrupt definition. 0 Transaction with address only transfer type causes regular interrupt. 1 Transaction with address only transfer type causes MCP interrupt.
30	DTO	Data time out. Data tenure time out interrupt definition. 0 Data tenure time out causes regular interrupt. 1 Data tenure time out causes MCP interrupt.
31	ATO	Address time out. Address tenure time out interrupt definition. 0 Address tenure time out causes regular interrupt. 1 Address tenure time out causes MCP interrupt.

7.2.5 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 7-5](#) shows the fields of AMR.

Offset 0x14

Access: Read/Write


Figure 7-5. Arbiter Mask Register (AMR)

[Table 7-6](#) describes AMR fields.

Table 7-6. AMR Field Descriptions

Bits	Name	Description
0–25	—	Write reserved, read = 0
26	ETEA	Transfer error. Detection of transfer error by one of the slaves interrupt mask bit. 0 Detection of transfer error by one of the slaves interrupt disabled. 1 Detection of transfer error by one of the slaves interrupt enabled.
27	RES	Reserved transfer type. Transaction with reserved transfer type interrupt mask bit. 0 Transaction with reserved transfer type interrupt disabled. 1 Transaction with reserved transfer type interrupt enabled.
28	ECW	External control word transfer type. Transaction with external control word transfer type interrupt mask bit. 0 Transaction with external control word transfer type interrupt disabled. 1 Transaction with external control word transfer type interrupt enabled.
29	AO	Address only transfer type. Transaction with address only transfer type interrupt mask bit. 0 Transaction with address only transfer type interrupt disabled. 1 Transaction with address only transfer type interrupt enabled.

Table 7-6. AMR Field Descriptions (continued)

Bits	Name	Description
30	DTO	Data time out. Data tenure time out interrupt mask bit. 0 Data tenure time out interrupt disabled. 1 Data tenure time out interrupt enabled.
31	ATO	Address time out. Address tenure time out interrupt mask bit. 0 Address tenure time out interrupt disabled. 1 Address tenure time out interrupt enabled.

7.2.6 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 7.2.3, “Arbiter Event Register \(AER\),”](#) for more information. AEATR is cleared only by power-on reset. The attributes of the first error event are stored. Note that this means that AEATR does not change its value when AER is not clear. As AEATR is not affected by soft or hard reset, software can read this register and determine the cause of the bus failure, even if the failure caused a deadlock situation. For more information, see [Section 7.4.2, “Error Handling Sequence.”](#)

Figure 7-6 shows the fields of AEATR.

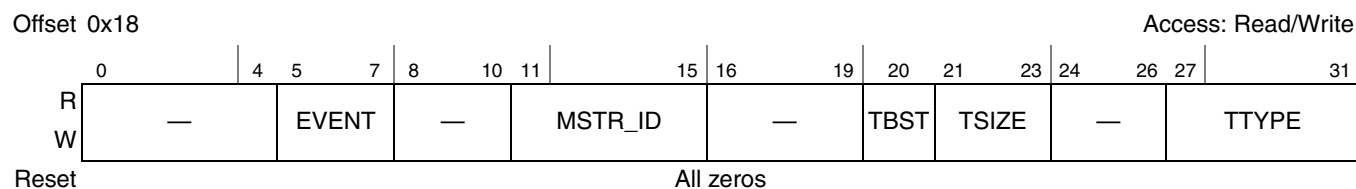


Figure 7-6. Arbiter Event Attributes Register (AEATR)

Table 7-7 describes AEATR fields.

Table 7-7. AEATR Field Descriptions

Bits	Name	Description
0–4	—	Write reserved, read = 0
5–7	EVENT	Event type. 000 Address time out 001 Data time out 010 Address only transfer type 011 External control word transfer type 100 Reserved transfer type 101 Transfer error 11c Reserved
8–10	—	Write reserved, read = 0

Table 7-7. AEATR Field Descriptions (continued)

Bits	Name	Description																																				
11–15	MSTR_ID	<p>Master Id.</p> <table border="0"> <tr> <td>00000 e300 core data transaction</td> <td>01010 JTAG</td> </tr> <tr> <td>00001 Reserved</td> <td>01011 Reserved</td> </tr> <tr> <td>00010 e300 core instruction fetch</td> <td>01100 eSDHC</td> </tr> <tr> <td>00011–00110 Reserved</td> <td>01101 PCI1</td> </tr> <tr> <td>00111 USB DR</td> <td>01110–01111 Reserved</td> </tr> <tr> <td>01000 Reserved</td> <td>10000 QE00</td> </tr> <tr> <td>01001 I2C (boot sequencer)</td> <td>10001 Reserved</td> </tr> <tr> <td></td> <td>10010–11110 Reserved</td> </tr> <tr> <td></td> <td>11111 DMA</td> </tr> </table> <p>Note: Master Id reflects the source of transaction and is used for debug purposes.</p>	00000 e300 core data transaction	01010 JTAG	00001 Reserved	01011 Reserved	00010 e300 core instruction fetch	01100 eSDHC	00011–00110 Reserved	01101 PCI1	00111 USB DR	01110–01111 Reserved	01000 Reserved	10000 QE00	01001 I2C (boot sequencer)	10001 Reserved		10010–11110 Reserved		11111 DMA																		
00000 e300 core data transaction	01010 JTAG																																					
00001 Reserved	01011 Reserved																																					
00010 e300 core instruction fetch	01100 eSDHC																																					
00011–00110 Reserved	01101 PCI1																																					
00111 USB DR	01110–01111 Reserved																																					
01000 Reserved	10000 QE00																																					
01001 I2C (boot sequencer)	10001 Reserved																																					
	10010–11110 Reserved																																					
	11111 DMA																																					
16–19	—	Write reserved, read = 0																																				
20	TBST	<p>Transfer burst.</p> <p>0 Burst transaction. Transfer size is greater than 8 bytes</p> <p>1 Single-beat transaction. Transfer size is up to 8 bytes</p>																																				
21–23	TSIZE	<p>Transfer size. Transfer size encoding depends on the value of the field TBST.</p> <table border="0"> <tr> <td colspan="2">TBST = 1:</td> <td colspan="2">TBST = 0:</td> </tr> <tr> <td>001 1 byte</td> <td></td> <td>000 16 bytes</td> <td></td> </tr> <tr> <td>010 2 bytes</td> <td></td> <td>001 24 bytes</td> <td></td> </tr> <tr> <td>011 3 bytes</td> <td></td> <td>010 32 bytes</td> <td></td> </tr> <tr> <td>100 4 bytes</td> <td></td> <td>011–111 Reserved</td> <td></td> </tr> <tr> <td>101 5 bytes</td> <td></td> <td></td> <td></td> </tr> <tr> <td>110 6 bytes</td> <td></td> <td></td> <td></td> </tr> <tr> <td>111 7 bytes</td> <td></td> <td></td> <td></td> </tr> <tr> <td>000 8 bytes</td> <td></td> <td></td> <td></td> </tr> </table>	TBST = 1:		TBST = 0:		001 1 byte		000 16 bytes		010 2 bytes		001 24 bytes		011 3 bytes		010 32 bytes		100 4 bytes		011–111 Reserved		101 5 bytes				110 6 bytes				111 7 bytes				000 8 bytes			
TBST = 1:		TBST = 0:																																				
001 1 byte		000 16 bytes																																				
010 2 bytes		001 24 bytes																																				
011 3 bytes		010 32 bytes																																				
100 4 bytes		011–111 Reserved																																				
101 5 bytes																																						
110 6 bytes																																						
111 7 bytes																																						
000 8 bytes																																						
24–26	—	Write reserved, read = 0																																				
27–31	TTYPE	<p>Transfer type.</p> <table border="0"> <tr> <td>00000 Address-only</td> <td>01111 Reserved</td> </tr> <tr> <td>00001 Address-only</td> <td>10000 Address-only</td> </tr> <tr> <td>00010 Single-beat or burst write</td> <td>1XX01 Reserved</td> </tr> <tr> <td>00011 Reserved</td> <td>10010 Single-beat write</td> </tr> <tr> <td>00100 Address-only</td> <td>1XX11 Reserved</td> </tr> <tr> <td>00101 Reserved</td> <td>10100 ecowx—Illegal single-beat write</td> </tr> <tr> <td>00110 Burst write</td> <td>10110 Reserved</td> </tr> <tr> <td>00111 Reserved</td> <td>11000 Address-only</td> </tr> <tr> <td>0100x Address-only</td> <td>11010 Single-beat or burst read</td> </tr> <tr> <td>0101x Single-beat or burst read</td> <td>11100 eciwx—Illegal single-beat read</td> </tr> <tr> <td>0110x Address-only</td> <td>11110 Burst read</td> </tr> <tr> <td>01110 Burst read</td> <td></td> </tr> </table>	00000 Address-only	01111 Reserved	00001 Address-only	10000 Address-only	00010 Single-beat or burst write	1XX01 Reserved	00011 Reserved	10010 Single-beat write	00100 Address-only	1XX11 Reserved	00101 Reserved	10100 ecowx —Illegal single-beat write	00110 Burst write	10110 Reserved	00111 Reserved	11000 Address-only	0100x Address-only	11010 Single-beat or burst read	0101x Single-beat or burst read	11100 eciwx —Illegal single-beat read	0110x Address-only	11110 Burst read	01110 Burst read													
00000 Address-only	01111 Reserved																																					
00001 Address-only	10000 Address-only																																					
00010 Single-beat or burst write	1XX01 Reserved																																					
00011 Reserved	10010 Single-beat write																																					
00100 Address-only	1XX11 Reserved																																					
00101 Reserved	10100 ecowx —Illegal single-beat write																																					
00110 Burst write	10110 Reserved																																					
00111 Reserved	11000 Address-only																																					
0100x Address-only	11010 Single-beat or burst read																																					
0101x Single-beat or burst read	11100 eciwx —Illegal single-beat read																																					
0110x Address-only	11110 Burst read																																					
01110 Burst read																																						

7.2.7 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 7.2.3, “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. For more information, see [Section 7.4.2, “Error Handling Sequence.”](#)

Figure 7-7 shows the fields of AEADR.



Figure 7-7. Arbiter Event Address Register (AEADR)

Table 7-8 describes AEADR fields.

Table 7-8. AEADR Field Descriptions

Bits	Name	Description
0–31	ADDR	Address of the event reported in AEATR register. See Section 7.2.6, “Arbiter Event Attributes Register (AEATR),” for more information.

7.2.8 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 7-8](#) shows the fields of AERR.

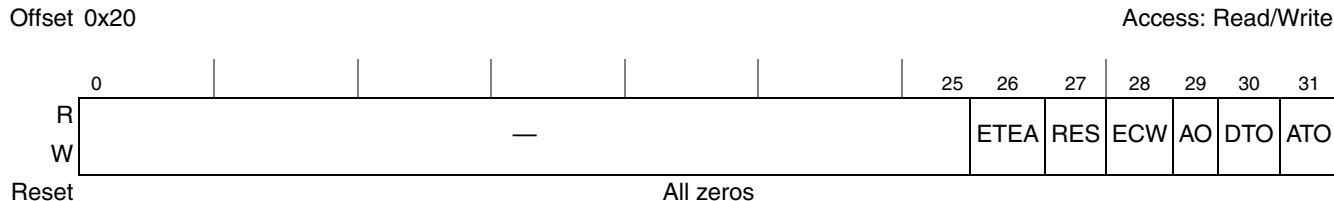


Figure 7-8. Arbiter Event Response Register (AERR)

Table 7-9 describes AERR field.

Table 7-9. AERR Field Descriptions

Bits	Name	Description
0–25	—	Write reserved, read = 0
26	ETEA	Transfer error. Detection of transfer error by one of the slaves event response. 0 Detection of transfer error by one of the slaves causes interrupt. 1 Detection of transfer error by one of the slaves causes reset request.
27	RES	Reserved transfer type. Transaction with reserved transfer type interrupt definition. 0 Transaction with reserved transfer type causes interrupt. 1 Transaction with reserved transfer type causes reset request.

Table 7-9. AERR Field Descriptions

Bits	Name	Description
28	ECW	External control word transfer type. Transaction with external control word transfer type interrupt definition. 0 Transaction with external control word transfer type causes interrupt. 1 Transaction with external control word transfer type causes reset request.
29	AO	Address only transfer type. Transaction with address only transfer type interrupt definition. 0 Transaction with address only transfer type causes interrupt. 1 Transaction with address only transfer type causes reset request.
30	DTO	Data time out. Data tenure time out interrupt definition. 0 Data tenure time out causes interrupt. 1 Data tenure time out causes reset request.
31	ATO	Address time out. Address tenure time out interrupt definition. 0 Address tenure time out causes interrupt. 1 Address tenure time out causes reset request.

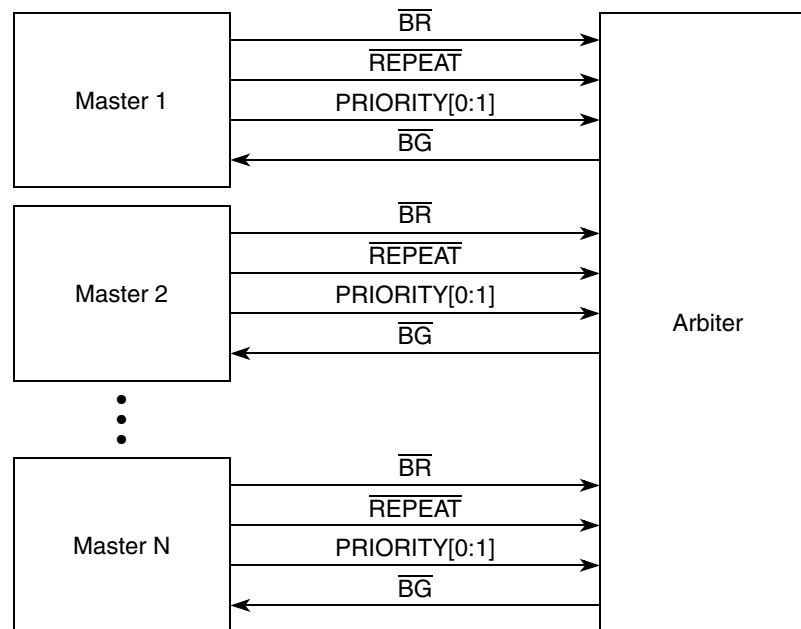
7.3 Functional Description

The following sections describe arbitration policy and bus error detection.

7.3.1 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 7-9 shows the interface signals between the arbiter and masters that are involved in address bus arbitration.


Figure 7-9. Address Bus Arbitration

A master has to acquire address bus ownership before it starts any transaction. The master asserts its own bus request signal along with the arbitration attribute signals $\overline{\text{REPEAT}}$ and $\text{PRIORITY}[0:1]$. The arbiter later asserts the corresponding address bus grant signal to the requesting master depending on the system states and arbitration scheme. See [Section 7.3.1.1, “Address Bus Arbitration with \$\text{PRIORITY}\[0:1\]\$,”](#) for details on arbitration scheme. When address bus grant is received the master can start the address tenure.

7.3.1.1 Address Bus Arbitration with $\text{PRIORITY}[0:1]$

Whenever a master asserts its bus request to acquire address bus ownership, it can drive its $\text{PRIORITY}[0:1]$ signals to indicate request priority. The master would be served sooner because of its higher priority level. The arbiter takes this extra information into consideration in order to yield better service for a higher priority request than a lower priority request. Therefore, the arbiter operates according to the following priority based arbitration scheme:

1. For every priority level a fair arbitration scheme is used (a simple round robin scheme)
2. For every priority level other than 0, one place is reserved as a place holder for lower level arbitration rings.
3. Each master can change its priority level at any time.

[Figure 7-10](#) shows an example of priority-based arbitration algorithm with four priority levels. In this example, if all masters request the bus continuously, the following order of bus grants occurs with the specific bandwidth:

- M6 gets $\frac{1}{2}$ of the bus bandwidth
- M4 and M5 each gets $\frac{1}{6}$ of the bus bandwidth
- M0 and M3 each gets $\frac{1}{18}$ of the bus bandwidth
- M1 and M2 each gets $\frac{1}{36}$ of the bus bandwidth

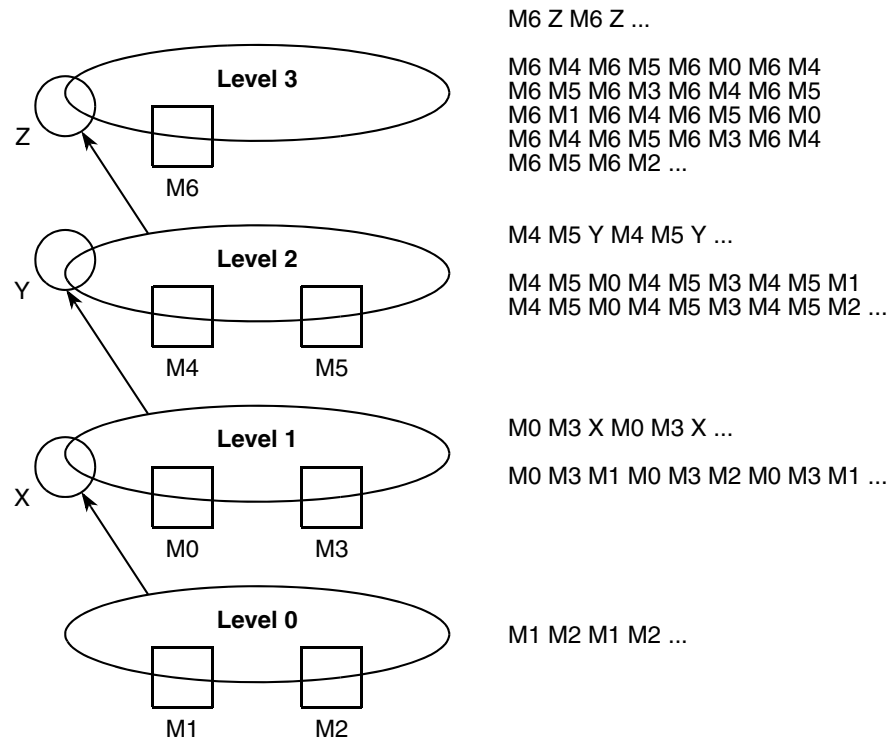


Figure 7-10. An Example of Priority-Based Arbitration Algorithm

NOTE

See each bus master's chapter and [Section 6.3.2.4, "System Priority and Configuration Register \(SPCR\),"](#) for more details about priority programming.

7.3.1.2 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.

7.3.1.3 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.

7.3.1.4 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 7.2.1, “Arbiter Configuration Register \(ACR\),”](#) for more details about ACR[APARK] and ACR[PARKM].

7.3.1.5 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.

7.3.2 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

7.3.2.1 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.

7.3.2.2 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.

7.3.2.3 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.

7.3.2.4 Address Only Transaction Type

Table 7-10 shows transaction types, which are defined as address only:

Table 7-10. 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.

7.3.2.5 Reserved Transaction Type

Table 7-11 shows transaction types defined as reserved.

Table 7-11. 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{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.

7.3.2.6 Illegal (eciwx/ecowx) Transaction Type

Table 7-12 shows transaction types defined as illegal.

Table 7-12. 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{AACK} .
2. Starts data tenure and ends data tenure by asserting \overline{TEA} .
3. Reports on the event in AER[ECW].
4. Issues reset request, MCP or regular interrupt according to AERR[ECW] and AIDR[ECW], if enabled by AMR[ECW].
5. Updates transaction attributes and address of AEATR and AEADR for the first error event.

For more information, see the following sections:

- [Section 7.2.3, “Arbiter Event Register \(AER\)”](#)
- [Section 7.2.4, “Arbiter Interrupt Definition Register \(AIDR\)”](#)
- [Section 7.2.5, “Arbiter Mask Register \(AMR\)”](#)
- [Section 7.2.6, “Arbiter Event Attributes Register \(AEATR\)”](#)
- [Section 7.2.7, “Arbiter Event Address Register \(AEADR\)”](#)

- [Section 7.2.8, “Arbiter Event Response Register \(AERR\)”](#)

7.4 Initialization/Applications Information

The following sections describe the initialization and error handling sequences for the arbiter.

7.4.1 Initialization Sequence

The following initialization sequence is recommended:

1. Write to ACR to configure pipeline depth, address bus parking mode, global maximum repeat count.
2. Write to AERR defines whether different error events cause a reset request or an interrupt.
3. Write to AIDR defines the kind of interrupt (regular or MCP) caused by each error event. Note that this is necessary only if interrupts are enabled and AERR defines error events to cause interrupt.
4. Write to AMR to enable interrupts.
5. Write to ATR to set the ATO and DTO timers. Note that this is only necessary if the required timers are less than the maximum value (which is default).

7.4.2 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 8

e300 Processor Core Overview

This chapter provides an overview of features for the embedded microprocessors in the e300 core family, which are PowerPC microprocessors built on Power Architecture technology. Throughout this chapter, the terms ‘e300 core’, ‘core’, and ‘processor’ are used interchangeably. The term, ‘e300c3’ is used when describing an implementation-specific feature or when a difference exists between different configurations. The term ‘e300’ is used when describing a feature that pertains to the family of e300 processors. The MPC8309 uses e300c3 core.

8.1 Overview

This section describes the details of the e300 core, provides a block diagram showing the major functional units, and briefly describes how these units interact. For additional information, see *e300 Power Architecture Core Family Reference Manual*.

The e300 core is a low-power implementation of this microprocessor family of reduced instruction set computing (RISC) microprocessors. The core implements the 32-bit portion of the PowerPC architecture, which defines 32-bit effective addresses, integer data types of 8, 16, and 32 bits, and floating-point data types of 32 and 64 bits.

The core is a superscalar processor that can issue and retire as many as three instructions per clock cycle. Instructions can execute out of program order for increased performance; however, the core makes completion appear sequential.

The e300 core integrates independent execution units including: an integer unit (IU) a floating-point unit (FPU), a branch processing unit (BPU), a load/store unit (LSU), and a system register unit (SRU). The e300c3 integrates an additional integer unit for a total of two IUs. The ability to execute instructions in parallel and the use of simple instructions with rapid execution times yield high efficiency and throughput for e300-core-based systems. Most integer instructions execute in one clock cycle. The additional IUs along with enhanced multipliers in the e300c3 improve multiply instructions to a maximum two-cycle latency, a significant improvement from previous processors. In the e300c3 core, the FPU is pipelined so a single-precision multiply-add instruction can be issued and completed every clock cycle. The e300c3 core provide hardware support for all single- and double-precision floating-point operations for most value representations and all rounding modes.

Figure 8-1 shows a block diagram of the e300c3 core. Note that the e300c3 supports floating-point operations and includes two integer units.

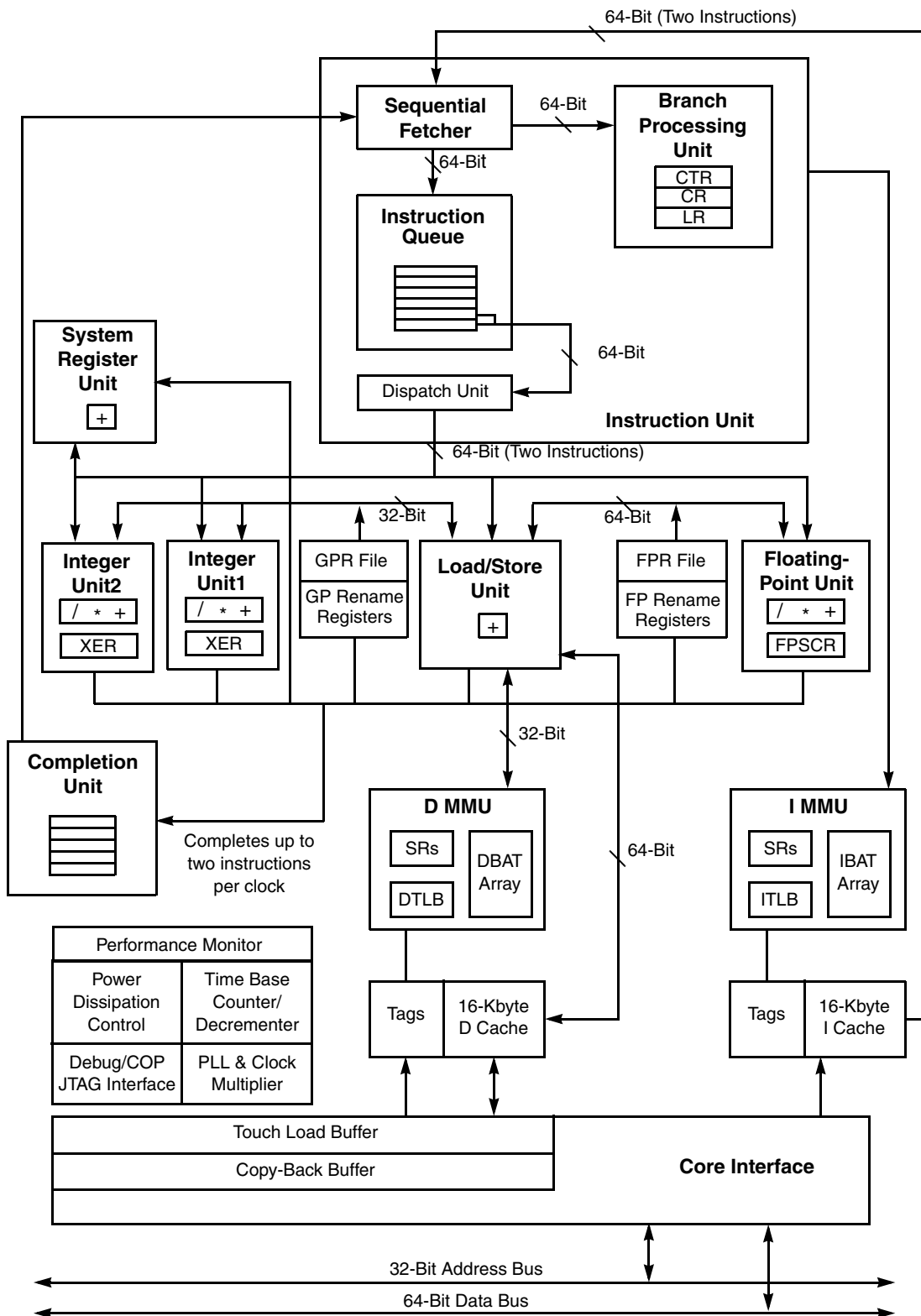


Figure 8-1. e300c3 Core Block Diagram

The e300c3 includes 16-Kbyte, four way set-associative instruction and data caches. The MMUs contain 64-entry, two-way, set-associative, data and instruction translation lookaside buffers (DTLB and ITLB) that provide support for demand-paged, virtual-memory, address translation, and variable-sized block translation. The TLBs use a least recently used (LRU) replacement algorithm and the caches use a pseudo least recently used algorithm (PLRU).

The core also supports block address translation through the use of two independent instruction and data block address translation (IBAT and DBAT) arrays, each containing eight pairs of BATs, an increase from four pairs of each type of BATs in the G2 core. This increase provides more flexibility in protecting accesses and providing translation on a segment, block, or page basis for memory accesses and I/O accesses. Effective addresses are compared simultaneously with all eight entries in the BAT array during block translation. In accordance with the PowerPC architecture, if an effective address hits in both the TLB and BAT array, the BAT translation takes priority.

As part of the coherent system bus (CSB), the e300 core has a 64-bit data bus and a 32-bit address bus. During normal operation, the e300 core provides a three-state (modified, exclusive, and invalid) coherency protocol which is a compatible subset of a four-state (modified/exclusive/shared/invalid) MESI protocol. However, the e300 data cache contains a programmable MESI extension that supports the shared cache coherency state (similar to other PowerPC processors). Both protocols operate coherently in systems that contain four-state caches. Although MESI is supported by the e300 core, it is not implemented on the MPC8309. The core also supports single-beat and burst data transfers for memory accesses and supports memory-mapped I/O operations.

The true little-endian mode is another enhanced capability of the e300 core. Unlike the PowerPC little-endian mode (which manipulates only the address bits), no longer supported on the e300, the true little-endian mode actually operates on true little-endian instructions and data from memory.

The critical interrupt is an additional interrupt in the e300 core and has higher priority order than the system management interrupt. Also, debug features are improved in the e300. Additional SPRG interrupt handling registers are provided for enhancing flexibility for the operating system.

The e300c3 include a performance monitor facility that provides the ability to monitor and count predefined events such as core clocks, misses in the instruction cache, data cache, types of instructions dispatched, mispredicted branches, and other occurrences. The count of such events (which may be an approximation) can be used to trigger the performance monitor interrupt. [Section 8.1.7.5, “Core Performance Monitor,”](#) describes the operation of the performance monitor diagnostic tool.

8.1.1 Features

This section describes the major features of the e300 core:

- High-performance, superscalar microprocessor core
 - As many as three instructions issued and retired per clock (two instructions plus one branch instruction)
 - As many as five instructions in execution per clock
 - Single-cycle execution for most instructions
 - Pipelined floating-point unit (FPU) for all single- and double-precision operations

- Independent execution units and two register files
 - Branch processing unit (BPU) featuring static branch prediction
 - Two 32-bit integer units (IU) in the e300c3
 - FPU based on the IEEE Std 754™ for both single- and double-precision operations
 - Load/store unit (LSU) for data transfer between data-cache and general-purpose registers (GPRs) and floating-point registers (FPRs)
 - System register unit (SRU) that executes condition register (CR), special-purpose register (SPR), and integer add/compare instructions. Add/compare instructions are also executed in the IUs.
 - Thirty-two 32-bit GPRs for integer operands
 - Thirty-two 64-bit FPRs for single- or double-precision operands
- High instruction and data throughput
 - Zero-cycle branch capability (branch folding)
 - Programmable static branch prediction on unresolved conditional branches
 - Two integer units with enhanced multipliers in the e300c3 for increased integer instruction throughput and a maximum two-cycle latency for multiply instructions
 - Instruction fetch unit capable of fetching two instructions per clock from the instruction cache
 - A six-entry instruction queue (IQ) that provides lookahead capability
 - Independent pipelines with feed-forwarding that reduces data dependencies in hardware
 - 16-Kbyte, four-way set-associative instruction and data caches on the e300c3
 - Cache write-back or write-through operation programmable on a per-page or per-block basis
 - Features for instruction and data cache locking and protection
 - BPU that performs CR lookahead operations
 - Address translation facilities for 4-Kbyte page size, variable block size, and 256-Mbyte segment size
 - A 64-entry, two-way, set-associative ITLB and DTLB
 - Eight-entry data and instruction BAT arrays providing 128-Kbyte to 256-Mbyte blocks
 - Software table search operations and updates supported through fast trap mechanism
 - 52-bit virtual address; 32-bit physical address
- Facilities for enhanced system performance
 - A 64-bit split-transaction internal data bus interface to the coherent system bus (CSB) with burst transfers
 - Support for one-level address pipelining on the CSB interface
 - True little-endian mode for compatibility with other true little-endian devices
 - Critical interrupt support
 - Hardware support for misaligned little-endian accesses
 - Configurable processor bus frequency multipliers as defined in the *MPC8309 PowerQUICC II Pro Integrated Communications Processor Hardware Specification*

- Integrated power management
 - Internal processor/bus clock multiplier ratios
 - Three power-saving modes: doze, nap, and sleep
 - Automatic dynamic power reduction when internal functional units are idle
- In-system testability and debugging features through JTAG boundary-scan capability

Features specific to the e300 core not present on the G2 processors follow:

- Enhancements to the register set
 - The e300 core has one more HID0 bit than the G2:
 - The enable cache parity checking (ECPE) bit, HID0[1], gives the e300 core the ability to enable the taking of a machine check interrupt based on the detection of a cache parity error
- Enhancements to cache implementation
 - 16-Kbyte, four-way set-associative instruction and data caches on the e300c3.
 - Full parity checking is performed on both instruction and data cache memory arrays
 - Lockable L1 instruction and data caches—entire cache or on a per-way basis up to 3 of 4 ways on the e300c3
 - New **icbt** instruction supports initialization of instruction cache
 - Data cache supports four-state MESI coherency protocol (not implemented on MPC8309)
 - The instruction cache is blocked only until the critical load completes (hit under reloads allowed)
 - Instruction cancel mechanism improves utilization of instruction cache by supporting hits-under-cancels and misses-under-cancels.
 - The critical double word is simultaneously written to the cache and forwarded to the requesting unit, thus minimizing stalls due to load delays.
 - Data cache queue sharing makes cast-outs and snoop pushes more efficient
 - Provides for an optional data cache operation broadcast feature (enabled by HID0[ABE]) that allows for coherent system management. All of the data cache control instructions, except **dcbz** (**dcbi**, **dcbf**, and **dcbst**) require that HID0[ABE] be enabled to broadcast.
 - Instruction fetch burst feature allows all instruction fetches from caching-inhibited space to be performed on the bus as burst transactions
- Interrupts
 - The e300 core offers hardware support for misaligned little-endian accesses. Little-endian load/store accesses that are not on a word boundary, except for strings and multiples, generate interrupts under the same circumstances as big-endian accesses.
 - The e300 core supports true little-endian mode to minimize the impact on software porting from true little-endian systems.
 - An input interrupt signal, \overline{cint} , is provided to trigger the critical interrupt exception on the e300 core. The pm_event_in input signal can be used by the performance monitor counters to trigger an interrupt upon overflow on the e300c3 .
- Bus clock—PLL configuration signals include seven signals for settings and control: *pll_cfg[0:6]*.

- Debug features
 - Breakpoint status recorded in DBCR and IBCR control registers
 - Two signals for the debug interface: *stopped* and *ext_halt*
 - Performance monitor registers for system analysis in the e300c3

Figure 8-1 provides a block diagram of the e300 core that shows how the execution units—IU, FPU, BPU, LSU, and SRU—operate independently and in parallel. It should be noted that this is a conceptual diagram and does not attempt to show how these features are physically implemented on the device.

The e300 core provides address translation and protection facilities, including an ITLB, 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 8.1.2, “Instruction Unit,” and Section 8.1.5.1, “Memory Management Units (MMUs).”

8.1.2 Instruction Unit

As shown in Figure 8-1, the e300 core instruction unit, which contains a fetch unit, instruction queue, dispatch unit, and BPU, provides centralized control of instruction flow to the execution units. The instruction unit determines the address of the next instruction to be fetched based on information from the sequential fetcher and from the BPU.

The instruction unit fetches the instructions from the instruction cache into the instruction queue. The BPU receives branch instructions from the fetcher and uses static branch prediction to allow fetching from a predicted instruction stream while a conditional branch is evaluated. The BPU folds out for unconditional branch instructions and conditional branch instructions unaffected by instructions in the execution pipeline.

Instructions issued beyond a predicted branch cannot complete execution until the branch is resolved, preserving the programming model of sequential execution. If any of these are branch instructions, they are decoded but not issued. Instructions to be executed by the FPU, IU, LSU, and SRU are issued and allowed to progress up to the register write-back stage. Write-back is allowed when a correctly predicted branch is resolved, and execution continues along the predicted path.

If branch prediction is incorrect, the instruction unit flushes all predicted path instructions, and instructions are issued from the correct path.

8.1.2.1 Instruction Queue and Dispatch Unit

The instruction queue (IQ), shown in Figure 8-1, holds as many as six instructions and loads up to two instructions from the instruction unit during a single cycle. The instruction fetch unit continuously loads as many instructions as space in the IQ allows. Instructions are dispatched to their respective execution units from the dispatch unit at a maximum rate of two instructions per cycle. Dispatching is facilitated to the IUs, FPU, LSU, and SRU by the provision of a reservation station at each unit. The dispatch unit performs source and destination register dependency checking, determines dispatch serializations, and inhibits subsequent instruction dispatching as required.

For a more detailed overview of instruction dispatch, see Section 8.4.6, “Instruction Timing.”

8.1.2.2 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 (**bclrx**) instruction. The CTR contains the branch target address for the Branch Conditional to Count Register (**bcctrx**) 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.

8.1.3 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.

8.1.3.1 Integer Unit (IU)

The IU executes all integer instructions. The IU executes one integer instruction at a time, performing computations with its arithmetic logic unit (ALU), multiplier, divider, and XER register. Most integer instructions are single-cycle instructions. The 32 GPRs hold integer operands. Stalls due to contention for GPRs are minimized by the automatic allocation of rename registers. The core writes the contents of the rename registers to the appropriate GPR when integer instructions are retired by the completion unit. The e300c3 provides two integer units for greater integer instruction throughput along with enhanced multipliers in each IU for faster multiply-instruction execution.

8.1.3.2 Floating-Point Unit (FPU)

The FPU contains a single-precision multiply-add array and the floating-point status and control register (FPSCR). The multiply-add array allows the core to efficiently implement multiply and multiply-add operations. The FPU is pipelined so that single- and double-precision instructions can be issued back-to-back. The 32 FPRs are provided to support floating-point operations. Stalls due to contention for FPRs are minimized by the automatic allocation of rename registers. The core writes the contents of the rename registers to the appropriate FPR when floating-point instructions are retired by the completion unit.

The e300c3 core supports all floating-point data types based on the IEEE 754 standard (normalized, denormalized, NaN, zero, and infinity) in hardware, eliminating the latency incurred by software interrupt routines.

8.1.3.3 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.

8.1.3.4 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.

8.1.4 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.

8.1.5 Memory Subsystem Support

The core provides separate instruction and data caches and MMUs. The core also provides an efficient processor bus interface to facilitate access to main memory and other bus subsystems. The memory subsystem support functions are described in the following sections.

8.1.5.1 Memory Management Units (MMUs)

The core MMUs support up to 4 Petabytes (2^{52}) of virtual memory and 4 Gigabytes (2^{32}) of physical memory (referred to as real memory in the architecture specification) for instruction and data. The MMUs also control access privileges for these spaces on block and page granularities. Referenced and changed status is maintained by the processor for each page to assist implementation of a demand-paged virtual memory system. Note that software assistance is required for the device to maintain reference and changed status. A key bit is implemented to provide information about memory protection violations prior to page table search operations.

The LSU calculates effective addresses for data loads and stores, performs data alignment to and from cache memory, and provides the sequencing for load and store string and multiple word instructions. The instruction unit calculates effective addresses for instruction fetching.

After an EA is generated, its higher-order bits are translated by the appropriate MMU into physical address bits. The lower-order EA bits are the same on the physical address which are directed to the on-chip cache and formed the index into a four-way set-associative tag array. After translating the address, the MMU passes the higher-order physical address bits to the cache and the cache lookup completes. For caching-inhibited accesses or accesses that miss in the cache, the untranslated lower-order address bits are concatenated with the translated higher-order address bits; the resulting 32-bit physical address is then used by the memory unit and the core interface to access external memory.

The MMU also directs the address translation and enforces the protection hierarchy programmed by the operating system in relation to the supervisor/user privilege level of the access and in relation to whether the access is a load or store.

For instruction fetches, the IMMU looks for the address in the ITLB and in the IBAT array. If an address hits both, the IBAT array translation is used. Data accesses cause a lookup in the DTLB and DBAT array. In most cases, the translation is in a TLB and the physical address bits are available to the on-chip cache.

The e300 core implements four more IBAT and four more DBAT entries than the G2.

When the EA misses in the TLBs, the core provides hardware assistance for software to perform a search of the translation tables in memory. The hardware assist consists of the following features:

- Automatic storage of the missed effective address in IMISS and DMISS
- Automatic generation of the primary and secondary hashed real addresses of the page-table entry group (PTEG), which are readable from the HASH1 and HASH2 register locations.
The HASH data is generated from the contents of the IMISS or DMISS register. The register that is selected depends on the miss (instruction or data) that was last acknowledged.
- Automatic generation of the first word of the page table entry (PTE) of the tables being searched
- A real page address (RPA) register that matches the format of the lower word of the PTE
- TLB access instructions (**tlbli** and **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.

For more information about memory management for the core, see [Section 8.4.5.2, “Implementation-Specific Memory Management.”](#)

8.1.5.2 Cache Units

The e300c3 provides 16-Kbyte, four-way set-associative instruction and data caches. The cache block is 32 bytes long. The caches adhere to a write-back policy, but the e300 core allows control of cacheability, write policy, and memory coherency at the page and block levels. The caches use a pseudo LRU replacement policy.

As shown in [Figure 8-1](#), 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 re-arbitrate 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.

8.1.6 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.

8.1.7 System Support Functions

The e300 core implements several support functions that include power management, time base/decrementer registers for system timing tasks, a JTAG (based on IEEE Std 1149.1™) interface, hardware debug, and a phase-locked loop (PLL) clock multiplier. These system support functions are described in the following sections.

8.1.7.1 Power Management

The e300 core provides four power modes, selectable by setting the appropriate control bits in the machine state register (MSR) and the hardware implementation register 0 (HID0). When entering into a power mode other than full-power, the core requests entry via a *qreq* signal and enters 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 automatically enters a low-power state without affecting performance, software execution, or external hardware.
- Doze
All the functional units of the e300 core are disabled except for the time base/decrementer registers and the bus snooping logic. When the processor is in doze mode, an external asynchronous interrupt, system management interrupt, decrementer interrupt, hard or soft reset, or machine check brings the e300 core into the full-power state. The core in doze mode maintains the PLL in a fully-powered state and locked to the system external clock input (*sysclk*), so a transition to the full-power state takes only a few processor clock cycles.
- Nap
The nap mode further reduces power consumption by disabling bus snooping, leaving only the time base register and the PLL in a powered state. The core returns to the full-power state on receipt of an external asynchronous interrupt, system management interrupt, decrementer interrupt, hard or soft reset, or machine check input (*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 *mcp* signal after the time required to relock the PLL.

8.1.7.2 Time Base/Decrementer

The time base is a 64-bit register (accessed as two 32-bit registers) that is incremented once every four bus clock cycles; external control of the time base is provided through the time base/decrementer clock base enable (*tben*) signal. The decrementer is a 32-bit register that generates a decrementer interrupt after a programmable delay. The contents of the decrementer register are decremented once every four bus clock cycles, and the decrementer interrupt is generated as the count passes through zero.

8.1.7.3 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 8.4.8, “Debug Features,”](#) for more information.

8.1.7.4 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).

8.1.7.5 Core Performance Monitor

The performance monitor provides the ability to count predefined events and processor clocks associated with particular operations, such as cache misses, mispredicted branches, or the number of cycles an execution unit stalls. The count of such events can be used to trigger the performance monitor interrupt.

The performance monitor can be used to do the following:

- Improve system performance by monitoring software execution and then recoding algorithms for more efficiency. For example, memory hierarchy behavior can be monitored and analyzed to optimize task scheduling or data distribution algorithms.
- Characterize processors in environments not easily characterized by benchmarking.
- Help system developers bring up and debug their systems.

The performance monitor uses the following resources:

- The performance monitor mark bit in the MSR (MSR[PMM]). This bit controls which programs are monitored.
- The move to/from performance monitor registers (PMR) instructions, **mtpmr** and **mfpmr**.
- The external core input, *pm_event_in*.
- PMRs

- The performance monitor counter registers (PMC0–PMC3) are 32-bit counters used to count software-selectable events. Each counter counts up to 128 events. UPMC0–UPMC3 provide user-level read access to these registers. They are identified in [Table 8-2](#).
- The performance monitor global control register (PMGC0) controls the counting of performance monitor events. It takes priority over all other performance monitor control registers. UPMGC0 provides user-level read access to PMGC0.
- The performance monitor local control registers (PMLCa0–PMLCa3) control each individual performance monitor counter. Each counter has a corresponding PMLCa register. UPMLCa0–UPMLCa3 provide user-level read access to PMLCa0–PMLCa3).
- The performance monitor interrupt is assigned to interrupt vector 0x0F00.

Software communication with the performance monitor is achieved through PMRs rather than SPRs. The PMRs are used for enabling conditions that can trigger the performance monitor interrupt.

8.2 e300 Processor and System Version Numbers

[Table 8-1](#) lists the revision codes in the processor version register (PVR) and the system version register (SVR) to the revision level marked on the device. These registers can be accessed as SPRs through the e300 core (see [Figure 8-2](#)).

Table 8-1. Device Revision Level Cross-Reference

Device	Revision	Processor Version Register (PVR)	System Version Register (SVR)
MPC8309	1.1	0x8085_0020	0x8110_0011
MPC8309	1.0	0x8085_0020	0x8110_0010

8.3 PowerPC Architecture Implementation

The PowerPC architecture consists of the following layers, and adherence to the PowerPC architecture can be measured in terms of which of the following levels of the architecture is implemented:

- User instruction set architecture (UISA)
 - Defines the base user-level instruction set, user-level registers, data types, floating-point interrupt model, memory models for a uniprocessor environment, and programming model for a uniprocessor environment.
- Virtual environment architecture (VEA)
 - Describes the memory model for a multiprocessor environment, defines cache control instructions, and describes other aspects of virtual environments. Implementations that conform to the VEA also adhere to the UISA but may not necessarily adhere to the OEA.
- Operating environment architecture (OEA)
 - Defines the memory management model, supervisor-level registers, synchronization requirements, and interrupt model. Implementations that conform to the OEA also adhere to the UISA and VEA.

The PowerPC architecture allows a wide range of designs for such features as cache and core interface implementations.

8.4 Implementation-Specific Information

This section describes the PowerPC architecture in general and specific details about the implementation of the e300 core as a low-power, 32-bit member of this PowerPC core family. The main topics addressed are as follows:

- [Section 8.4.1, “Register Model,”](#) describes the registers for the operating environment architecture common among e300 cores that implement the PowerPC architecture and describes the programming model. It also describes the additional registers that are unique to the core.
- [Section 8.4.2, “Instruction Set and Addressing Modes,”](#) describes the PowerPC instruction set and addressing modes for the OEA, and defines and describes the instructions implemented in the core.
- [Section 8.4.3, “Cache Implementation,”](#) describes the cache model that is defined generally for cores that implement the PowerPC architecture by the VEA. It also provides specific details about the e300 core cache implementation.
- [Section 8.4.4, “Interrupt Model,”](#) describes the interrupt model of the OEA and the differences in the core interrupt model.
- [Section 8.4.5, “Memory Management,”](#) describes generally the conventions for memory management among these cores. This section also describes the core implementation of the 32-bit PowerPC memory management specification.
- [Section 8.4.6, “Instruction Timing,”](#) provides a general description of the instruction timing provided by the superscalar, parallel execution supported by the PowerPC architecture and the e300 core.
- [Section 8.1.6, “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 8.1.1, “Features.”](#)

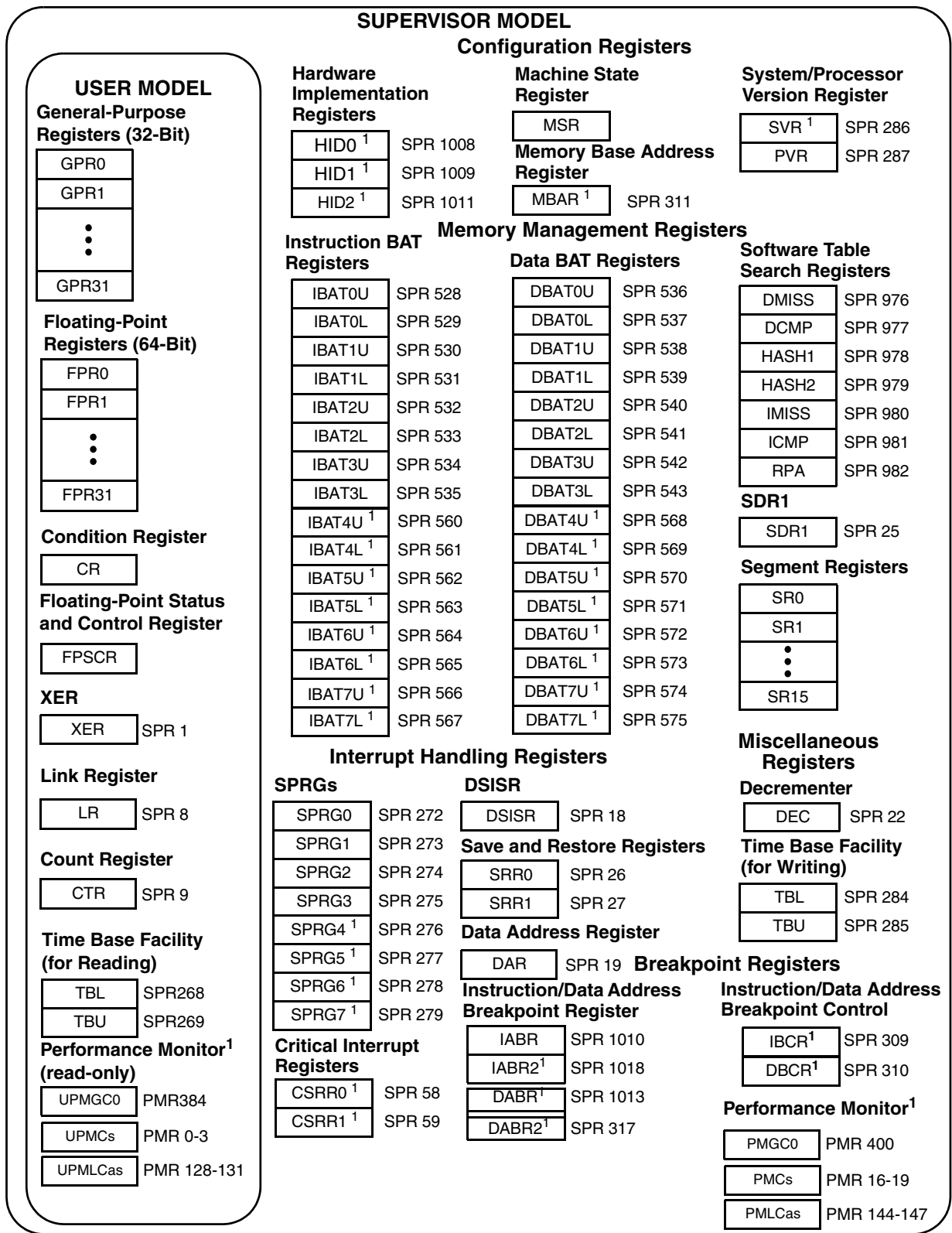
8.4.1 Register Model

The PowerPC architecture defines register-to-register operations for most computational instructions. Source operands for these instructions are accessed from the registers or are provided as immediate values embedded in the instruction opcode. The three-register instruction format allows specification of a target register distinct from the two-source operands. Load and store instructions transfer data between registers and memory.

The e300 core has two levels of privilege: supervisor mode of operation (typically used by the operating system) and user mode of operation (used by the application software). The programming models incorporate 32 GPRs, 32 FPRs, special-purpose registers (SPRs), and several miscellaneous registers. Each core also has its own unique set of hardware implementation (HID) registers.

Having access to privileged instructions, registers, and other resources allows the operating system to control the application environment (providing virtual memory and protecting operating system and critical machine resources). Instructions that control the state of the e300 core, the address translation mechanism, and supervisor registers can be executed only when the core is operating in supervisor mode.

Figure 8-2 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 8-2. e300 Programming Model—Registers

The following sections describe the e300-core-implementation-specific features as they apply to registers.

8.4.1.1 UISA Registers

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

8.4.1.1.1 General-Purpose Registers (GPRs)

The PowerPC architecture defines 32 user-level GPRs that are 32 bits wide in 32-bit cores. The GPRs serve as the data source or destination for all integer instructions.

8.4.1.1.2 Floating-Point Registers (FPRs)

The PowerPC architecture also defines 32 user-level, 64-bit FPRs. The FPRs serve as the data source or destination for floating-point instructions. These registers can contain data objects of either single- or double-precision floating-point formats.

8.4.1.1.3 Condition Register (CR)

The CR is a 32-bit user-level register that provides a mechanism for testing and branching. It consists of eight 4-bit fields that reflect the results of certain operations, such as move, integer and floating-point comparisons, arithmetic, and logical operations.

8.4.1.1.4 Floating-Point Status and Control Register (FPSCR)

The user-level FPSCR contains all floating-point exception signal bits, exception summary bits, exception enable bits, and rounding control bits needed for compliance with the IEEE 754 standard.

8.4.1.1.5 User-Level SPRs

The PowerPC architecture defines numerous special purpose registers that serve a variety of functions, such as providing controls, indicating status, configuring the core, and performing special operations. During normal execution, a program can access the registers, as shown in [Figure 8-2](#), 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.

8.4.1.2 VEA Registers

The VEA introduces the time base facility (TB) for reading. The TB is a 64-bit register pair whose contents are incremented once every four core input clock cycles. The TB consists of two 32-bit registers—time base upper (TBU) and time base lower (TBL). Note that the time base registers are read-only in user state.

8.4.1.3 OEA Registers

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

8.4.1.3.1 Machine State Register (MSR)

The MSR is a supervisor-level register that defines the state of the core. The contents of this register are saved when an interrupt is taken, and restored when the interrupt handling completes. A critical interrupt is taken in the e300 core when the \overline{cint} signal is asserted and MSR[CE] is set. The e300 core implements the MSR as a 32-bit register.

Table 8-2 shows the bit definitions for MSR.

Table 8-2. 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.
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.

Table 8-2. MSR Bit Descriptions (continued)

Bits	Name	Description
16	EE	External interrupt enable 0 The processor ignores external interrupts, system management interrupts, and decremter interrupts. 1 The processor is enabled to take an external interrupt, system management interrupt, or decremter interrupt.
17	PR	Privilege level 0 The processor can execute both user- and supervisor-level instructions 1 The processor can only execute user-level instructions
18	FP	Floating-point available 0 The processor prevents dispatch of floating-point instructions, including floating-point loads, stores, and moves. 1 The processor can execute floating-point instructions and can take floating-point enabled exception type program interrupts.
19	ME	Machine check enable 0 Machine check interrupts are disabled 1 Machine check interrupts are enabled
20	FE0	Floating-point exception mode 0
21	SE	Single-step trace enable 0 The processor executes instructions normally 1 The processor generates a trace interrupt upon the successful completion of the next instruction
22	BE	Branch trace enable 0 The processor executes branch instructions normally 1 The processor generates a trace interrupt upon the successful completion of a branch instruction
23	FE1	Floating-point exception mode 1
24	CE	Critical interrupt enable 0 Critical interrupts disabled 1 Critical interrupts enabled; critical interrupt and 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, <i>nnnn</i> is the offset of the interrupt. 0 Interrupts are vectored to the physical address 0x000n_nnnn 1 Interrupts are vectored to the physical address 0xFFFFn_nnnn
26	IR	Instruction address translation 0 Instruction address translation is disabled 1 Instruction address translation is enabled
27	DR	Data address translation 0 Data address translation is disabled 1 Data address translation is enabled
28	—	Reserved. Full function.
29	PMM	Performance monitor mark bit (e300c3). System software can set PMM when a marked process is running to enable statistics to be gathered only during the execution of the marked process. MSR[PR] and MSR[PMM] together define a state that the processor (supervisor or user) and the process (marked or unmarked) may be in at any time. If this state matches an individual state specified in the PMLCan, the state for which monitoring is enabled, counting is enabled.

Table 8-2. MSR Bit Descriptions (continued)

Bits	Name	Description
30	RI	Recoverable interrupt (for system reset and machine check interrupts) 0 Interrupt is not recoverable 1 Interrupt is recoverable
31	LE	Little-endian mode enable 0 The processor runs in big-endian mode 1 The processor runs in little-endian mode.

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

8.4.1.3.2 Segment Registers (SRs)

For memory management, 32-bit processors implement sixteen 32-bit SRs. To speed access, the core implements the SRs as two arrays: a main array, for data memory accesses, and a shadow array, for instruction memory accesses. Loading a segment entry with the move to segment register (**mtsr**) instruction loads both arrays.

8.4.1.3.3 Supervisor-Level SPRs

The e300 core, like the G2_LE core, has additional supervisor-level SPRs, which are shown in [Figure 8-3](#). Two critical interrupt SPRs (CSRR0 and CSRR1), eight SPRGs (SPRG0–SPRG7), eight pairs of instruction BATs (IBAT0–IBAT7), eight pairs of data BATs (DBAT0–DBAT7), one system version register (SVR), one system memory base address (MBAR), one instruction address breakpoint control (IBCR), one data address breakpoint control (DBCR), a new instruction breakpoint register (IABR2), and two data address breakpoint registers (DABR and DABR2) are integrated into the core.

The following list discusses the supervisor-level SPRs.

- The DSISR defines the cause of data access and alignment interrupts. The cause of a DSI interrupt for a data breakpoint (match with DABR and DABR2) can be determined by the value of the DSISR[DABR] bit (bit 9).
- The data address register (DAR) holds the address of an access after an alignment or DSI interrupt. For example, it contains the address of the breakpoint match condition.
- The decremter register (DEC) is a 32-bit decrementing counter that provides a mechanism for causing a decremter interrupt after a programmable delay.
- SDR1 specifies the page table format used in virtual-to-physical address translation for pages. (Note that physical address is referred to as ‘real address’ in the architecture specification.)
- The machine status save/restore register 0 (SRR0) is used for saving the address of the instruction that caused the interrupt, and the address to return to when a Return from Interrupt (**rfi**) instruction is executed.
- The machine status save/restore register 1 (SRR1) is used to save machine status on interrupts and to restore machine status when an **rfi** instruction is executed.
- The SPRG0–SPRG7 registers are provided for operating system use. They reduce the latency that may be incurred in the saving of registers to memory while in a handler. Note that the e300 implements four more SPRGs than the G2 (SPRG0–SPRG3).

- The time base register (TB) is a 64-bit register that maintains the time of day and operates interval timers. It consists of two 32-bit fields: time base upper (TBU) and time base lower (TBL).
- The processor version register (PVR) is a read-only register that identifies the version (model) and revision level of the processor. See [Table 8-9](#) for the version and revision level of the PVR for the e300 processor core.
- The PowerPC architecture defines 16 block address translation (BAT) registers. The e300 core includes a total of eight pairs of DBAT and eight pairs of IBAT registers. See [Figure 8-2](#) 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 8-3 shows the bit definitions for HID0.

Table 8-3. e300 HID0 Bit Descriptions

Bits	Name	Function
0	EMCP	Enable \overline{mcp} . The purpose of this bit is to mask out machine check interrupts caused by assertion of \overline{mcp} , similar to how MSR[EE] can mask external interrupts. 0 Masks \overline{mcp} . Asserting \overline{mcp} does not generate a machine check interrupt or a checkstop. 1 Asserting \overline{mcp} causes checkstop if MSR[ME] = 0 or a machine check interrupt if ME = 1
1	ECPE	Enable cache parity errors. 0 Disables instruction and data cache parity error reporting 1 Allows a detected cache parity error to cause a machine check interrupt if MSR[ME] = 1 or a checkstop if MSR[ME] = 0
2	EBA	Enable $\overline{ap_in[0:3]}$ and \overline{ape} for address parity checking. 0 Disables address parity checking during a snoop operation 1 Allows an address parity error during snoop operations to cause a checkstop if MSR[ME] = 0 or a machine check interrupt if MSR[ME] = 1
3	EBD	Enable \overline{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 \overline{hreset} to configure clk_out . See Table 8-4 for settings.
5	—	Reserved, should be cleared
6	ECLK	clk_out output enable. Used in conjunction with HID0[SBCLK] and the \overline{hreset} signal to configure clk_out . See Table 8-4 for settings.
7	PAR	Disable precharge of $\overline{artry_out}$ 0 Precharge of $\overline{artry_out}$ enabled 1 Alters bus protocol slightly by preventing the processor from driving $\overline{artry_out}$ to high (negated) state. If this is done, the integrated device must restore the signals to the high state.
8	DOZE	Doze mode enable. Operates in conjunction with MSR[POW]. 0 Doze mode disabled 1 Doze mode enabled. Doze mode is invoked by setting MSR[POW] while this bit is set. In doze mode, the PLL, time base, and snooping remain active.
9	NAP	Nap mode enable. Operates in conjunction with MSR[POW]. 0 Nap mode disabled 1 Nap mode enabled. Nap mode is invoked by setting MSR[POW] while this bit is set. In nap mode, the PLL and time base remain active.
10	SLEEP	Sleep mode enable. Operates in conjunction with MSR[POW]. 0 Sleep mode disabled 1 Sleep mode enabled. Sleep mode is invoked by setting MSR[POW] while this bit is set. \overline{qreq} is asserted to indicate that the processor is ready to enter sleep mode. If the system logic determines that the processor may enter sleep mode, the quiesce acknowledge signal, \overline{qack} , is asserted back to the processor. Once \overline{qack} assertion is detected, the processor enters sleep mode after several processor clocks. At this point, the system logic may turn off the PLL by first configuring $pll_cfg[0:6]$ to PLL bypass mode, then disabling $sysclk$.

Table 8-3. e300 HID0 Bit Descriptions (continued)

Bits	Name	Function
11	DPM	Dynamic power management enable 0 Dynamic power management is disabled 1 Functional units enter a low-power mode automatically if the unit is idle. This does not affect operational performance and is transparent to software or any external hardware.
12–15	—	Reserved, should be cleared.
16	ICE	Instruction cache enable 0 The instruction cache is neither accessed nor updated. All pages are accessed as if they were marked cache-inhibited (WIM = x1x). Potential cache accesses from the bus (snoop and cache operations) are ignored. In the disabled state for the L1 caches, the cache tag state bits are ignored and all instruction fetches are propagated to the coherent system bus (CSB) as single-beat transactions. For those transactions, however, \bar{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, \bar{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, \bar{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, \bar{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 8-3. e300 HID0 Bit Descriptions (continued)

Bits	Name	Function
21	DCFI	Data cache Flash invalidate 0 The data cache is not invalidated. The bit is cleared when the invalidation operation begins (usually the next cycle after the write operation to the register). The data cache must be enabled for the invalidation to occur. 1 An invalidate operation is issued that marks the state of each data cache block as invalid without writing back modified cache blocks to memory. Cache access is blocked during this time. Bus accesses to the cache are signaled as a miss during invalidate-all operations. Setting DCFI clears all the valid bits of the blocks and the PLRU bits to point to way L0 of each set. For the e300 core, the proper use of the ICFI and DCFI bits is to set and clear them with two consecutive 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	DECAREN	Decrementer auto reload 0 Normal operation. 1 Decrementer loads last mtdec value for precise periodic interrupt.
26	—	Reserved, should be cleared.
27	FBIOB	Force branch indirect on the bus 0 Register indirect branch targets are fetched normally 1 Forces register indirect branch targets to be fetched externally
28	ABE	Address broadcast enable. Controls whether certain address-only operations (such as cache operations) are broadcast on the bus. 0 Address-only operations affect only local caches and are not broadcast 1 Address-only operations are broadcast on the bus Affected instructions are 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 dcbst instructions are enabled 1 The dcbt and dcbst instructions are no-oped internal to the e300 core

Table 8-4 shows how HID0[ECLK] and HID0[SBCLK] are used to configure the *clk_out* signal.

Table 8-4. Using HID0[ECLK] and HID0[SBCLK] to Configure *clk_out*

<i>hreset</i>	ECLK	SBCLK	<i>clk_out</i>
Asserted	x	x	Bus clock (small pulse for every rising edge of sysclk)
Negated	0	0	Clock output off
	0	1	Core clock/2
	1	0	Core clock
	1	1	Bus clock

Table 8-5 shows the bit definitions for HID1

Table 8-5. HID1 Bit Descriptions

Bits	Name	Description
0	PC0	PLL configuration bit 0 (read-only)
1	PC1	PLL configuration bit 1 (read-only)
2	PC2	PLL configuration bit 2 (read-only)
3	PC3	PLL configuration bit 3 (read-only)
4	PC4	PLL configuration bit 4 (read-only)
5	PC5	PLL configuration bit 5 (read-only)
6	PC6	PLL configuration bit 6 (read-only)
7–31	—	Reserved, should be cleared

Note: The clock configuration bits reflect the state of the *pll_cfg[0:6]* signals.

Table 8-6 shows the bit definitions for HID2.

Table 8-6. e300HID2 Bit Descriptions

Bits	Name	Description
0–3	—	Reserved, should be cleared.
4	LET	True little-endian. This bit enables true little-endian mode operation for instruction and data accesses. This bit is set to reflect the state of the <i>tle</i> signal at the negation of <i>hreset</i> . This bit is used in conjunction with MSR[LE] to determine the endian mode of operation. 0 No function 1 True little-endian mode, when MSR[LE] = 1 Changing the value of this bit during normal operation is not recommended
5	IFEB	Instruction fetch burst extension. This bit enables the instruction fetch burst extension. 0 Instruction fetch burst extension disabled 1 Instruction fetch burst extension enabled
6	—	Reserved, should be cleared.
7	MESISTATE	MESI state enable. This bit enables the four-state MESI cache coherency protocol. 0 MESI disabled. The data cache uses a three-state MEI coherency protocol. 1 MESI enabled. The data cache uses a four-state MESI protocol.
8	IFEC	Instruction fetch cancel extension. This bit enables the instruction fetch cancel extension. 0 Instruction fetch cancel extension disabled 1 Instruction fetch cancel extension enabled
9	EBQS	Enable BIU queue sharing. This bit enables data cache queue sharing. 0 Data cache queue sharing disabled 1 Data cache queue sharing enabled
10	EBPX	Enable BIU pipeline extension. This bit enables the bus interface unit pipeline extension. 0 BIU pipeline extension disabled; 1 level pipeline 1 BIU pipeline extension enabled; 1-1/2 level pipeline
11–12	—	Reserved for e300c1, should be cleared.

Table 8-6. e300HID2 Bit Descriptions (continued)

Bits	Name	Description
11	ELRW	Enable weighted LRU. This bit enables the use of an adjusted (weighted) LRU. 0 Normal operation. 1 The dcbt, dcbst, and dcbz instructions use an adjusted (weighted) LRU such that they always select and replace the lowest unlocked way in the data cache.
12	NOKS	No kill for snoop. This bit enables the forcing of kill-type snoops to flush data instead of killing it. 0 Normal operation. 1 Forces write-with-kill snoops to flush instead of kill (snoop can never kill data).
13	HBE	High BAT enable. Regardless of the setting of HID2[HBE], these BATs are accessible by mf spr and mt spr . 0 IBAT[4–7] and DBAT[4–7] are disabled 1 IBAT[4–7] and DBAT[4–7] are enabled
14–15	—	Reserved, should be cleared.
16–18	IWLCK[0–2]	Instruction cache way-lock. Useful for locking blocks of instructions into the instruction cache for time-critical applications that require deterministic behavior. 000 no ways locked 001 way 0 locked 010 way 0 through way 1 locked 011 way 0 through way 2 locked 100 way 0 through way 2 locked in e300c3. 101 way 0 through way 2 locked in e300c3. 110 way 0 through way 2 locked in e300c3. 111 way 0 through way 2 locked in e300c3. Setting HID0[ILOCK] locks all ways.
19	ICWP	Instruction cache way protection. Used to protect locked ways in the instruction cache from being invalidated. 0 Instruction cache way protection disabled 1 Instruction cache way protection enabled
20–23	—	Reserved, should be cleared.
24–26	DWLCK[0–2]	Data cache way-lock. Useful for locking blocks of data into the data cache for time-critical applications where deterministic behavior is required. 000 no ways locked 001 way 0 locked 010 way 0 through way 1 locked 011 way 0 through way 2 locked 100 way 0 through way 2 locked in e300c3. 101 way 0 through way 2 locked in e300c3. 110 way 0 through way 2 locked in e300c3. 111 way 0 through way 2 locked in e300c3. Setting HID0[DLOCK] locks all ways.
27–31	—	Reserved, should be cleared.

8.4.2 Instruction Set and Addressing Modes

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

8.4.2.1 PowerPC Instruction Set and Addressing Modes

All PowerPC instructions are encoded as single-word (32-bit) opcodes. Instruction formats are consistent among all instruction types, permitting efficient decoding to occur in parallel with operand accesses. This fixed instruction length and consistent format simplifies instruction pipelining.

The PowerPC instructions are divided into the following categories:

- Integer instructions (includes computational and logical instructions)
 - Integer arithmetic instructions
 - Integer compare instructions
 - Integer logical instructions
 - Integer rotate and shift instructions
- Floating-point instructions (includes floating-point computational instructions and instructions that affect the FPSCR)
 - Floating-point arithmetic instructions
 - Floating-point multiply/add instructions
 - Floating-point rounding and conversion instructions
 - Floating-point compare instructions
 - Floating-point status and control instructions
- Load/store instructions (includes integer and floating-point load and store instructions)
 - Integer load and store instructions
 - Integer load and store multiple instructions
 - Floating-point load and store
 - Primitives used to construct atomic memory operations (**lwarx** and **stwcx.** instructions)
- Flow control instructions (includes branching instructions, condition register logical instructions, trap instructions, and other instructions that affect the instruction flow)
 - Branch and trap instructions
 - Condition register logical instructions
- Processor control instructions (includes instructions used for synchronizing memory accesses and managing caches, TLBs, and the segment registers)
 - Move to/from SPR instructions
 - Move to/from MSR
 - Move to/from PMR
 - Synchronize
 - Instruction synchronize
- Memory control instructions (includes instructions that provide control of caches, TLBs, and segment registers)
 - Supervisor-level cache management instructions
 - Translation lookaside buffer management instructions. Note that there are additional implementation-specific instructions.

- User-level cache instructions
- Segment register manipulation instructions

The e300 core implements the following instructions defined as optional by the PowerPC architecture:

- Floating select (**fsel**)
- Floating reciprocal estimate single-precision (**fres**)
- Floating reciprocal square root estimate (**frsqrte**)
- Store floating-point as integer word (**stfiwx**)

Note that this grouping of instructions does not indicate the execution unit that executes a particular instruction or group of instructions.

Integer instructions operate on byte, half-word, and word operands. Floating-point instructions operate on single-precision (one word) and double-precision (one double word) floating-point operands. The PowerPC architecture uses instructions that are 4 bytes long and word-aligned. It provides for byte, half-word, and word operand loads and stores between memory and a set of 32 GPRs. It also provides for word and double-word operand loads and stores between memory and a set of 32 FPRs.

Computational instructions do not modify memory. To use a memory operand in a computation and then modify the same or another memory location, the memory contents must be loaded into a register, modified, and then written back to the target location with distinct instructions.

The core follows the program flow when it is in the normal execution state. However, the flow of instructions can be interrupted directly by the execution of an instruction or by an asynchronous event. Either kind of interrupt may cause one of several components of the system software to be invoked.

8.4.2.2 Implementation-Specific Instruction Set

The e300 core instruction set is defined as follows:

- The core provides hardware support for all 32-bit PowerPC instructions.
- The core provides two implementation-specific instructions used for software table search operations following TLB misses:
 - Load Data TLB Entry (**tlbld**)
 - Load Instruction TLB Entry (**tlbli**)
- The core implements the following instruction 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 (**rfei**)
- 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 performance monitor instructions:
 - Move to Performance Monitor Register (**mtpmr**)
 - Move from Performance Monitor Register (**mfpmr**)

8.4.3 Cache Implementation

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

8.4.3.1 PowerPC Cache Characteristics

The PowerPC architecture does not define hardware aspects of cache implementations. The e300 core controls the following memory access modes on a page or block basis:

- Write-back/write-through mode
- Caching-inhibited mode
- Memory coherency

Note that in the core, a cache block is defined as eight words. The VEA defines cache management instructions that provide a means by which the application programmer can affect the cache contents.

8.4.3.2 Implementation-Specific Cache Organization

The e300c3 provides 16-Kbyte, four-way set-associative instruction and data caches. The caches are physically addressed, and the data cache can operate in either write-back or write-through mode as specified by the PowerPC architecture.

The data cache is configured as 128 sets of 4 blocks each on the e300c3. Each block consists of 32 bytes, 2 state bits, and an address tag. The two state bits implement the three-state MEI (modified/exclusive/invalid) protocol. Each block contains eight 32-bit words. Note that the PowerPC architecture defines the term ‘block’ as the cacheable unit. For the core, the block size is equivalent to a cache line. A block diagram of the data cache organization is shown in [Figure 8-3](#).

The instruction cache is configured as 128 sets of 4 blocks each on the e300c3. Each block consists of 32 bytes, an address tag, and a valid bit. The instruction cache may not be written to, except through a block fill operation. In the e300 core, the instruction cache is blocked only until the critical load completes. The e300 core supports instruction fetching from other instruction cache lines following the forwarding of the critical-first-double-word of a cache line load operation. Successive instruction fetches from the cache line being loaded are forwarded, and accesses to other instruction cache lines can proceed during the cache line load operation. The instruction cache is not snooped, and cache coherency must be maintained by software. A fast hardware invalidation capability is provided to support cache maintenance.

The e300c3 data cache is configured as 128 sets of four blocks per set. The organization of the data cache is shown in Figure 8-3.

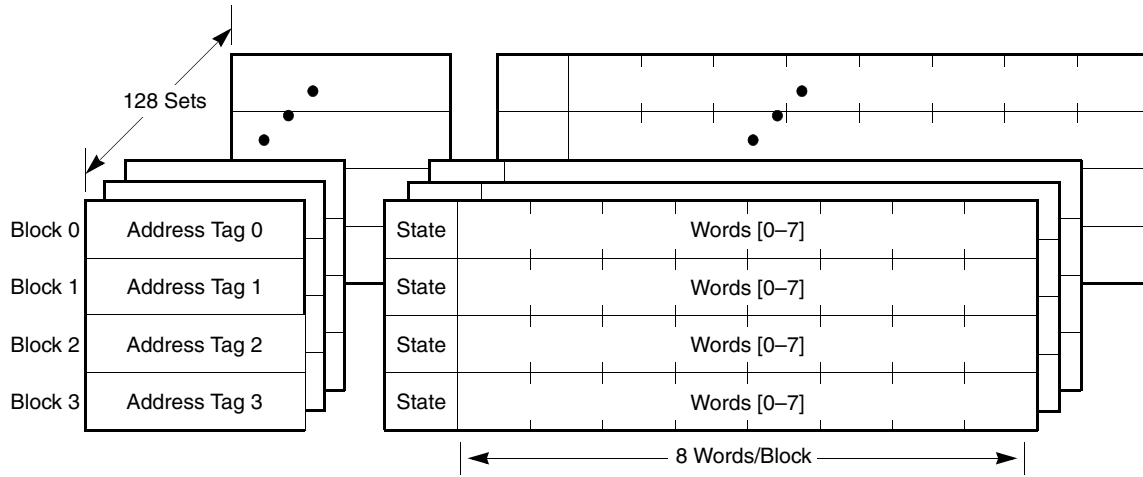


Figure 8-3. e300c3 Data Cache Organization

Each cache block contains eight contiguous words from memory that are loaded from an 8-word boundary (that is, bits A[27–31] of the effective addresses are zero); thus, a cache block never crosses a page boundary. Misaligned accesses across a page boundary can incur a performance penalty.

The e300 core cache blocks are loaded in four beats of 64 bits each on the 64-bit data bus. The burst load is performed as critical-double-word-first. The data cache is blocked to internal accesses until the load completes; the instruction cache allows sequential fetching during a cache block load. In the core, the critical-double-word is simultaneously written to the cache and forwarded to the requesting unit, thus minimizing stalls due to load delays.

To ensure coherency among caches in a multiprocessor (or multiple caching-device) implementation, the core implements the MEI protocol during normal operation of the data cache. The new data cache MESI extension supports the additional fourth cache coherency shared state for the data cache. To support this feature, the shared signal, *shd*, has been added to the bus interface. Although the MESI protocol is supported by the e300 core, it is not implemented on MPC8309. The following four states indicate the state of the cache block:

- **Modified**—The cache block is modified with respect to system memory; that is, data for this address is valid only in the cache and not in system memory.
- **Exclusive**—This cache block holds valid data that is identical to the data at this address in system memory. No other cache has this data.
- **Shared**—Only available if HID2[MESISTATE] register bit is set. The address block is valid in the cache and in at least one other cache. This block is always consistent with system memory. That is, the shared state is shared-unmodified; there is no shared-modified state. Although the MESI protocol is supported by the e300 core, it is not implemented on MPC8309.
- **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.

8.4.3.3 Instruction and Data Cache Way-Locking

The e300 core implements instruction and data cache way-locking, which guarantees that certain memory accesses hits in the cache. This provides deterministic access times for those accesses.

8.4.4 Interrupt Model

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

8.4.4.1 PowerPC Interrupt Model

The PowerPC interrupt mechanism allows the core to change to supervisor state as a result of external signals, errors, or unusual conditions arising in the execution of instructions. The conditions that can cause interrupts are called exceptions. When interrupts occur, information about the state of the core is saved to certain registers and the core begins execution at an address (interrupt vector) predetermined for each interrupt type. Interrupts are processed in supervisor mode.

Some interrupts, such as program interrupts, can be triggered by a broad range of exception conditions. Other interrupts, such as the decremter interrupt, have only a single exception condition. Although multiple exception conditions can map to a single interrupt vector, a more specific condition may be determined by examining a register associated with the interrupt—for example, the DSISR and the FPSCR. Additionally, some exception conditions can be explicitly enabled or disabled by software.

The PowerPC architecture requires that interrupts be handled in program order; therefore, although a particular implementation may recognize exception conditions out of order, they are presented strictly in order. When an instruction-caused interrupt is recognized, any unexecuted instructions that appear earlier in the instruction stream, including any that have not yet entered the execute stage, are required to complete before the interrupt is taken. Any interrupts caused by those instructions are handled first. Likewise, asynchronous, precise interrupts are recognized when they occur, but are not handled until the instruction currently in the completion stage successfully completes execution or generates an interrupt, and the completed store queue is emptied.

Unless a catastrophic condition causes a system reset or machine check interrupt, only one interrupt is handled at a time. If, for example, a single instruction encounters multiple interrupt conditions, those conditions are handled sequentially. After the interrupt handler completes, the instruction execution continues until the next interrupt condition is encountered. However, in many cases there is no attempt to re-execute the instruction. This method of recognizing and handling interrupts sequentially guarantees that interrupts are recoverable.

To prevent the program state from being lost due to a system reset, a machine check interrupt, or an instruction-caused interrupt in the interrupt handler, 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 completes execution before the interrupt is taken. Once the interrupt is processed, execution resumes at the address of the faulting instruction (or at an alternate address provided by the interrupt handler). When an interrupt is taken due to a trap or system call instruction, execution resumes at an address provided by the handler.
- **Synchronous, imprecise**
The PowerPC architecture defines two imprecise floating-point exception modes: recoverable and nonrecoverable. Even though the core provides a means to enable the imprecise modes, it implements these modes identically to the precise mode (that is, all enabled floating-point exceptions are always precise on the core).
- **Asynchronous, maskable**
The external system management interrupt (SMI) and decremter interrupts are maskable, asynchronous interrupts. When these interrupts occur, their handling is postponed until the next instruction and any of its associated interrupts complete execution. If there are no instructions in the execution units, the interrupt is taken immediately upon determination of the correct restart address (for loading SRR0).
- **Asynchronous, nonmaskable**
The system reset and the machine check interrupt are nonmaskable, asynchronous interrupts. They may not be recoverable, or they may provide a limited degree of recoverability. All interrupts report recoverability through MSR[RI].

8.4.4.2 Implementation-Specific Interrupt Model

As specified by the PowerPC architecture, all interrupts can be described as either precise or imprecise and either synchronous or asynchronous. Asynchronous interrupts (some of which are maskable) are caused by events external to the processor's execution; synchronous interrupts, which are all handled precisely by

the e300 core, are caused by instructions. A system management interrupt is an implementation-specific interrupt. The interrupt classes are shown in [Table 8-7](#).

Table 8-7. Interrupt Classifications

Synchronous/Asynchronous	Precise/Imprecise	Interrupt Type
Asynchronous, nonmaskable	Imprecise	Machine check System reset
Asynchronous, maskable	Precise	External interrupt Decrementer System management interrupt Critical interrupt
Synchronous	Precise	Instruction-caused interrupts

Although interrupts have other characteristics, such as whether they are maskable, the distinctions shown in [Table 8-7](#) define categories of interrupts that the core handles uniquely. Note that [Table 8-7](#) includes no synchronous, imprecise instructions. While the PowerPC architecture supports imprecise handling of floating-point exceptions, the core implements floating-point exception modes as precise.

The e300 core interrupts and exception conditions that cause them are listed in [Table 8-8](#).

Table 8-8. Exceptions and Interrupts

Interrupt Type	Vector Offset (hex)	Exception Conditions
Reserved	00000	—
System reset	00100	Caused by the assertion of either \overline{hreset} .
Machine check	00200	Caused by the assertion of the \overline{tea} signal during a data bus transaction, assertion of \overline{mcp} , an address or data parity error, or an instruction or data cache parity error. Note that the e300 has SRR1 register values that are different from the G2/G2_LE cores' when a machine check occurs.
DSI	00300	Determined by the bit settings in the DSISR, listed as follows: <ul style="list-style-type: none"> 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: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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.

Table 8-8. Exceptions and Interrupts (continued)

Interrupt Type	Vector Offset (hex)	Exception Conditions
External interrupt	00500	Caused when MSR[EE] = 1 and the \overline{int} signal is asserted.
Alignment	00600	Caused when the core cannot perform a memory access for any of the reasons described below: <ul style="list-style-type: none"> • 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. <ul style="list-style-type: none"> • 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 mfspir 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[0] changes from 0 to 1. This interrupt is enabled with MSR[EE].
Critical interrupt	00A00	Taken when \overline{cint} is asserted and MSR[CE] = 1.
Reserved	00B00–00BFF	—
System call	00C00	Occurs when a System Call (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.
Performance monitor	00F00	Caused when a configured PM counter using the pm_event_in to transition overflows.
Instruction translation miss	01000	Caused when the effective address for an instruction fetch cannot be translated by the ITLB.
Data load translation miss	01100	Caused when the effective address for a data load operation cannot be translated by the DTLB.

Table 8-8. Exceptions and Interrupts (continued)

Interrupt Type	Vector Offset (hex)	Exception Conditions
Data store translation miss	01200	Caused when the effective address for a data store operation cannot be translated by the DTLB, or when a DTLB hit occurs and the change bit in the PTE must be set due to a data store operation.
Instruction address breakpoint	01300	Occurs when the address (bits 0–29) in the IABR matches the next instruction to complete in the completion unit, and IABR[30] is set. Note that the e300 core also implements IABR2, which functions identically to IABR.
System management interrupt	01400	Caused when MSR[EE] = 1 and the \overline{smi} input signal is asserted.
Reserved	01500–02FFF	—

8.4.5 Memory Management

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

8.4.5.1 PowerPC Memory Management

The primary functions of the MMU are to translate logical (effective) addresses to physical addresses for memory accesses and to provide access protection on blocks and pages of memory.

The core generates two types of accesses that require address translation: instruction accesses and data accesses to memory generated by load and store instructions.

The PowerPC MMU and interrupt model support demand-paged virtual memory. Virtual memory management permits execution of programs larger than the size of physical memory; demand-paged implies that individual pages are loaded into physical memory from system memory only when they are first accessed by an executing program.

The hashed page table is a variable-sized data structure that defines the mapping between virtual page numbers and physical page numbers. The page table size is a power of two, and its starting address is a multiple of its size.

The page table contains a number of page-table entry groups (PTEGs). A PTEG contains eight page-table entries (PTEs) of 8 bytes each; therefore, each PTEG is 64 bytes long. PTEG addresses are entry points for table search operations.

Address translations are enabled by setting bits in the MSR. MSR[IR] enables instruction address translations, and MSR[DR] enables data address translations.

8.4.5.2 Implementation-Specific Memory Management

The instruction and data memory management units in the e300 core provide 4 Gbytes of logical address space accessible to supervisor and user programs with a 4-Kbyte page size and 256-Mbyte segment size. Block sizes range from 128 Kbytes to 256 Mbytes and are software selectable. In addition, the core uses

an interim 52-bit virtual address and hashed page tables for generating 32-bit physical addresses. The MMUs in the e300 core rely on the interrupt processing mechanism for the implementation of the paged virtual memory environment and for enforcing protection of designated memory areas.

Instruction and data TLBs provide address translation in parallel with the on-chip cache access, incurring no additional time penalty in the event of a TLB hit. A TLB is a cache of the most recently used page table entries. Software is responsible for maintaining the consistency of the TLB with memory. The core TLBs are 64-entry, two-way, set-associative caches that contain instruction and data address translations. The core provides hardware assist for software table search operations through the hashed page table on TLB misses. Supervisor software can invalidate TLB entries selectively.

For instructions and data that correspond to block address translation, the e300 core provides independent eight-entry BAT arrays. These entries define blocks that can vary from 128 Kbytes to 256 Mbytes. The BAT arrays are maintained by system software. HID2[HBE] is added to the e300 for enabling or disabling the four additional pairs of BAT registers. However, regardless of the setting of HID2[HBE], these BATs are accessible by **mfspr** and **mtspr**.

As specified by the PowerPC architecture, the hashed page table is a variable-sized data structure that defines the mapping between virtual page numbers and physical page numbers. The page table size is a power of two, and its starting address is a multiple of its size.

Also as specified by the PowerPC architecture, the page table contains a number of PTEGs. A PTEG contains 8 PTEs of 8 bytes each; therefore, each PTEG is 64 bytes long. PTEG addresses are entry points for table search operations.

8.4.6 Instruction Timing

The e300 core is a pipelined superscalar processor core. Because instruction processing is reduced into a series of stages, an instruction does not require all of the resources of an execution unit at the same time. For example, after an instruction completes the decode stage, it can pass on to the next stage, while the subsequent instruction can advance into the decode stage. This improves the throughput of the instruction flow. For example, it may take three cycles for a single floating-point instruction to execute, but if there are no stalls in the floating-point pipeline, a series of floating-point instructions can have a throughput of one instruction per cycle.

The core instruction pipeline has four major pipeline stages, described as follows:

- The fetch pipeline stage primarily involves retrieving instructions from the memory system and determining the location of the next instruction fetch. Additionally, if possible, the BPU decodes branches during the fetch stage and folds out branch instructions before the dispatch stage.
- The dispatch pipeline stage is responsible for decoding the instructions supplied by the instruction fetch stage and determining which of the instructions are eligible to be dispatched in the current cycle. In addition, the source operands of the instructions are read from the appropriate register file and dispatched with the instruction to the execute pipeline stage. At the end of the dispatch pipeline stage, the dispatched instructions and their operands are latched by the appropriate execution unit.
- In the execute pipeline stage, each execution unit with an instruction executes the selected instruction (perhaps over multiple cycles), writes the instruction's result into the appropriate rename register, and notifies the completion stage when the execution has finished. In the case of

an internal interrupt, the execution unit reports the interrupt to the completion/write-back pipeline stage and discontinues instruction execution until the interrupt is handled. The interrupt is not signaled until that instruction is the next to be completed. Execution of most floating-point instructions is pipelined within the FPU, allowing up to three instructions to execute in the FPU concurrently. The FPU pipeline stages are multiply, add, and round-convert. The LSU has two pipeline stages: the first stage, for effective address calculation and MMU translation, and the second, for accessing data in the cache.

- The complete/write-back pipeline stage maintains the correct architectural machine state and transfers the contents of the rename registers to the GPRs and FPRs as instructions are retired. If the completion logic detects an instruction causing an interrupt, all subsequent instructions are canceled, their execution results in rename registers are discarded, and instructions are fetched from the correct instruction stream.

A superscalar processor core issues multiple, independent instructions into multiple pipelines, allowing instructions to execute in parallel. The e300c1 core has independent execution units for: integer instructions, floating-point instructions, branch instructions, load/store instructions, and system register instructions. The e300c3 provides two IUs, which improves the throughput of integer instructions. The e300c3 provides two integer units for greater integer instruction throughput along with enhanced multipliers in each IU that reduce the multiply instruction latency to a maximum of two cycles. The IU and the FPU each have dedicated register files for maintaining operands (GPRs and FPRs, respectively), allowing integer and floating-point calculations to occur simultaneously without interference.

The core provides support for single-cycle store, and it provides an adder/comparator in the system register unit that allows the dispatch and execution of multiple integer add and compare instructions on each cycle.

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

8.4.7 Core Interface

The core interface is specific for each processor core implementation.

The MPC8309 contains an internal coherent system bus (CSB) that interfaces the processor core to the peripheral logic. In the case of the MPC8309, 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 8-4](#)). 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.

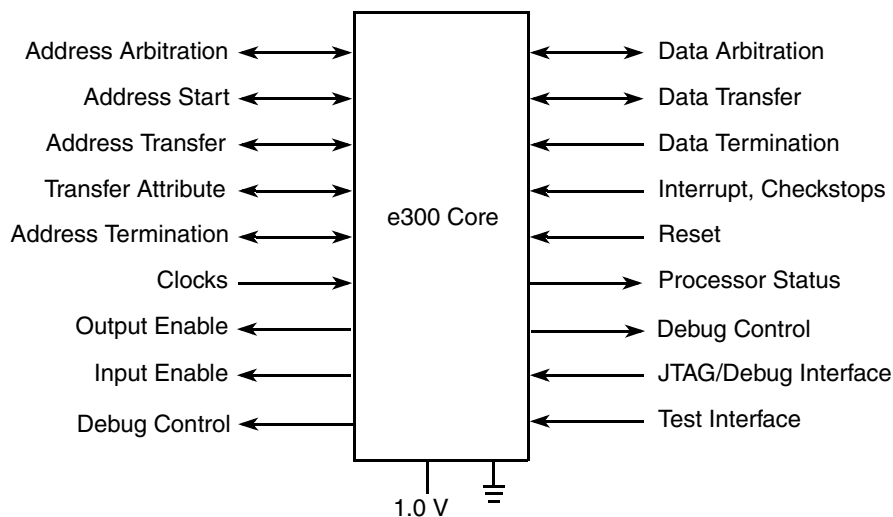


Figure 8-4. Core Interface

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.

8.4.7.1 Memory Accesses

The e300 core CSB is a 64-bit data bus.

With a 64-bit CSB, memory accesses allow transfer sizes of 8, 16, 24, 32, 40, 48, 56, or 64 bits in one bus clock cycle. Data transfers occur in either single-beat transactions or four-beat burst transactions. Single-beat transactions are caused by noncached accesses that access memory directly (that is, reads and writes when caching is disabled, caching-inhibited accesses, and stores in write-through mode). Four-beat burst transactions, which always transfer an entire cache block (32 bytes), are initiated when a line is read from or written to memory.

8.4.7.2 Signals

The e300 core signals are grouped as follows:

- Interrupts/Resets

These signals include the external interrupt signal (\overline{int}), critical interrupt signal (\overline{cint}), checkstop signals, performance monitor signal (pm_event_in) via the PM counters, and both soft reset and hard reset signals. They are used to interrupt and, under various conditions, to reset the core.

- JTAG/debug interface signals

The JTAG (based on the IEEE 1149.1 standard) interface and debug unit provides a serial interface to the system for performing monitoring and boundary tests. Two additional signals are added to the e300 core to allow observation of the internal clock state of the core ($stopped$) and to allow the external input to force the core into a halted state (ext_halt).

- Core status and control

These signals include the memory reservation signal, machine quiesce control signals, time base/decrementer clock base enable signal, and the $\overline{tlbisynd}$ 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.

8.4.8 Debug Features

Some new debug features are specific to the e300 core. Accesses to the debug facilities are available only in supervisor mode by using the **mtspr** and **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 .

8.4.8.1 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 $\overline{iabr2}$ and $\overline{dabr2}$ breakpoint signals are asserted after the AND condition is met (that is, both instruction breakpoints occurred or both data breakpoints occurred).
- When the $core_stopped$ pin is asserted, the e300 core has entered a stopped state and all internal clocking has stopped, indicating that a hardware debug event has occurred.
- The ext_halt input pin can be used to force the core into halted state. The halted state may be a hardstop, conditional upon the HARDSTOP condition being set through the JTAG/debug interface

8.5 Differences Between Cores

The e300 core has similar functionality to the G2_LE core. [Table 8-9](#) describes the differences between the G2_LE and the e300.

Table 8-9. Differences Between e300 and G2_LE Cores

e300 Core	G2_LE Core	Impact
New HID0 bits	—	The e300 core has a new HID0 bit defined to enable cache parity error reporting (ECPE).
New HID1 bits	—	The e300 core has new HID1 bits defined to extend the number of PLL configuration signals to seven (PC5, PC6).
New HID2 bits	—	The e300 core has new HID2 bits defined to support instruction fetch bursting (IFEB), MESI coherency protocol (MESI), instruction fetch cancels (IFEC), data cache queue sharing (EBQS), pipelining extension (EBPX), additional cache way locking (IWLCK and DWLCK), and instruction cache way protection (ICWP).
New PVR register value	—	The processor version register values differ.
New IBCR and DBCR bits	—	The e300 core has new IBCR[IABRSTAT, IABR2STAT] and DBCR[DABR1STAT, DABR2STAT] fields to provide instruction and data address breakpoint status.
—	16-Kbyte, four-way, set-associative, instruction and data caches	Some e300 cores may have different cache sizes than the G2_LE. For more information, see <i>e300 PowerPC Core Reference Manual</i> .
L1 cache parity	—	The e300 core supports parity for both instruction and data caches; the G2_LE does not support cache parity.
MEI or MESI coherency protocols	MEI protocol only	The e300 supports two coherency protocols: MEI and MESI; the G2_LE only supports the MEI protocol.
Instruction cancel extension	—	The e300 instruction cancel mechanism improves utilization of instruction cache by supporting ‘hits-under-cancels’ and ‘misses-under-cancels’; the G2_LE requires the cancel to complete before new instruction fetches can begin.
Instruction fetch bursts to caching-inhibited space	Single-beat instruction fetches to caching-inhibited space	The e300’s instruction fetch burst extension allows all caching-inhibited instruction fetches to be performed on the bus as burst transactions, even though the instructions are not cached. This improves performance for instruction space that is caching-inhibited, because up to eight instructions are returned with one bus operation. The G2_LE core must use single-beat instruction fetches for caching-inhibited space, returning only two instructions per bus operation.
Instruction cache way protection	—	The e300 core can protect locked ways in the instruction cache from invalidation; the G2_LE does not support instruction cache way protection.

Table 8-9. Differences Between e300 and G2_LE Cores (continued)

e300 Core	G2_LE Core	Impact
Data cache queue sharing	—	The e300 has a new data cache queue sharing extension that allows the two burst-write queues in the bus unit to be used interchangeably for cache replacements and snoop pushes. Thus, the data cache can support two outstanding cache replacements or two outstanding snoop push operations on the bus at any given time.
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.
1-½-level bus pipelining	1-level bus pipelining	For the e300, a new transaction can complete an address tenure when the previous transaction has been granted the data bus; for the G2_LE, a new transaction must wait until the previous data tenure has completed before completing its address tenure.
PowerPC little-endian not supported	PowerPC little-endian supported	PowerPC little-endian is not supported in the e300 core, although true little-endian is fully supported.
Data retry mode removed	Data retry mode available	\overline{drtry} and <i>drtrymode</i> is no longer supported on the e300 and future versions.
External control instructions removed	External control instructions available	The eciwx and ecowx instruction pair is not supported on the e300 core. These are optional instructions in the PowerPC architecture.
Reduced pin mode removed	Reduced pin mode available	Reduced pinout mode and the signal <i>redpinmode</i> is not supported in the e300 core.



Chapter 9

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.

9.1 Introduction

This chapter describes the IPIC interrupt protocol, various types of interrupt sources controlled by the IPIC unit, and the IPIC registers with some programming guidelines. The interrupt controller provides interrupt management that is responsible for receiving hardware-generated interrupts from different sources (both internal and external). It also prioritizes and delivers the interrupts to the CPU for servicing. The IPIC prioritizes and manages interrupts from the following controller units:

- DDR SDRAM memory controller (DDR2)
- Enhanced local bus memory controller (eLBC)
- PCI
- DMA Engine 1
- DMA Engine 2
- DUART communication module
- USB 2.0 dual role controller (USB DR)
- System bus arbiter (SBA)
- Periodic interval timer (PIT)
- Real time clock timer (RTC ALR and RTC SEC)
- Two global timer blocks
- Software watchdog timer (WDT)
- Two I²C controllers (I²C1 and I²C2)
- SPI
- Power management controller (PMC)
- General-purpose I/O controller (GPIO)
- QUICC Engine block
- External pins ($\overline{\text{IRQ0}}$, $\overline{\text{IRQ1}}$, $\overline{\text{IRQ2}}$, and $\overline{\text{IRQ3}}$)
- Enhanced secure digital host controller (SDHC)
- FlexCAN

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 9-1 shows the relationship of the various functional blocks and external signals of the device to the IPIC unit.

The IPIC receives interrupt request signals from the following two sources:

- External to the integrated device
- Internal to the integrated device

The unit selects the highest priority interrupt from all current interrupts and forwards it to the internal processor core, or off-chip for external servicing.

The IPIC also manages an internal non-maskable machine-check processor (\overline{mcp}) signal and the interrupt generated by the off-chip interrupt sources ($\overline{IRQ0}$, $\overline{IRQ1}$, $\overline{IRQ2}$, and $\overline{IRQ3}$).

The interrupt router of the IPIC monitors the outputs of the internal configuration registers. When the priority is highest in one of the received interrupt signals, the IPIC sets the corresponding bit in one of the following interrupt 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.

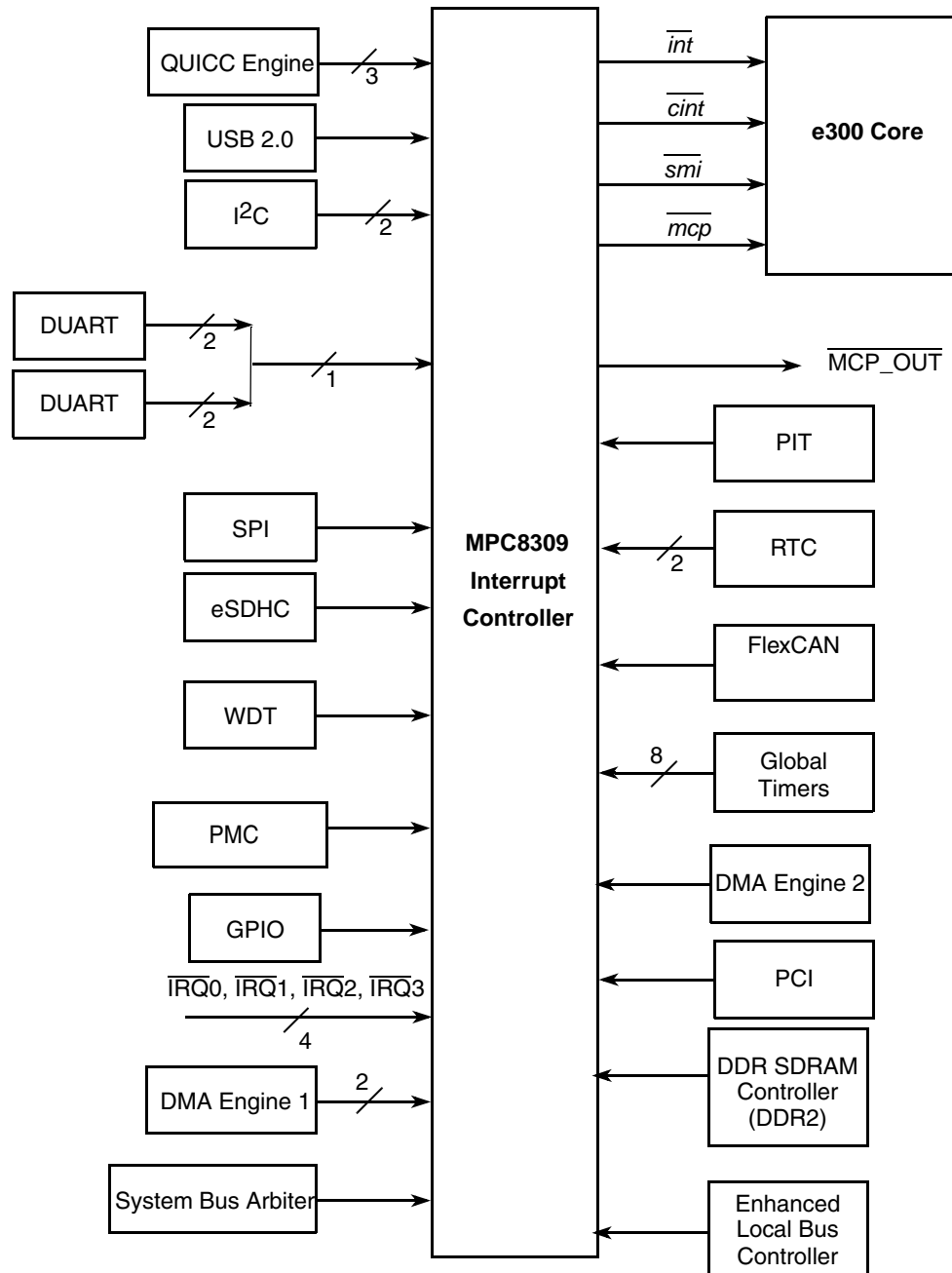


Figure 9-1. Interrupt Sources Block Diagram for MPC8309

The IPIC receives the following types of interrupts:

- External interrupt—triggered by the off-chip signals (\overline{IRQn}) listed in [Table 9-1](#)
- Internal interrupts—on-chip interrupts, triggered by the sources listed in [Table 9-8](#) and [Table 9-10](#)

- External and internal non-maskable machine check conditions, signaled by the sources listed in Table 9-24 through \overline{mcp}

The interrupt controller provides the ability to mask each interrupt source. Any source that can be caused by multiple events are also maskable.

When the IPIC receives an internal or external interrupt, its configuration register is checked to determine if it should be serviced as a normal interrupt by the processor core (through the \overline{int} signal) or if the incoming interrupt has been configured as a critical or system management interrupt, the IPIC completes the processing of the interrupt by asserting \overline{cint} or \overline{smi} to the core. The assertion of the \overline{cint} or \overline{smi} signal to the core causes the interrupt to be serviced as a critical or a system management interrupt, respectively.

9.2 Features

The IPIC unit implements the following features:

- Support for external and internal discrete vectorized interrupt sources
- Support for external and internal non-maskable machine check conditions, signaled by \overline{mcp}
- Support for dedicated external interrupts
- Programmable highest priority request (can be programmed to support a critical (\overline{cint}) or system management interrupt (\overline{smi}) type)
- Two programmable priority mixed groups of four on-chip and four external interrupt signals with two priority schemes for each group: grouped and spread
- Four programmable priority internal groups of eight on-chip interrupt signals with two priority schemes for each group: grouped and spread
- Two highest priority interrupts from each group can be programmed to support a critical or system management interrupt type
- External and internal interrupts directed to host processor
- Unique vector number for each interrupt source

9.3 Modes of Operation

The IPIC unit can operate in the core enable or core disable mode.

9.3.1 Core Enable Mode

In core enable mode, all internal interrupts are routed to and from the IPIC; the interrupts are sent to the e300C3 Power Architecture processor core. In this mode all machine check interrupts are gathered by the IPIC unit and sent to the e300C3 Power Architecture processor core.

9.3.2 Core Disable Mode

MPC8309 supports Core Disable Mode only for debug purposes. The e300C3 Power Architecture processor core may be put in Core Disable mode by RCW. In this mode, JTAG will have access to the registers for reads and writes.

9.4 External Signal Description

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

9.4.1 Overview

The device has 4 distinct external interrupt request input signals ($\overline{\text{IRQ0}}$, $\overline{\text{IRQ1}}$, $\overline{\text{IRQ2}}$, and $\overline{\text{IRQ3}}$). The IPIC interface signals are defined in [Table 9-1](#).

Table 9-1. IPIC Signal Properties

Name	Port	Function	I/O	Reset	Requires Pull Up
$\overline{\text{IRQ0}}$, $\overline{\text{IRQ1}}$, $\overline{\text{IRQ2}}$, and $\overline{\text{IRQ3}}$	$\overline{\text{IRQ0}}$, $\overline{\text{IRQ1}}$, $\overline{\text{IRQ2}}$, and $\overline{\text{IRQ3}}$	External interrupts	I	—	Yes
$\overline{\text{MCP_OUT}}$	$\overline{\text{MCP_OUT}}$	Interrupt request output	O	Z	Yes

9.4.2 Detailed Signal Descriptions

[Table 9-2](#) provides detailed descriptions of the external IPIC signals.

Table 9-2. IPIC External Signals—Detailed Signal Descriptions

Signal	I/O	Description
$\overline{\text{IRQ0}}$, $\overline{\text{IRQ1}}$, $\overline{\text{IRQ2}}$, and $\overline{\text{IRQ3}}$	I	Interrupt request 0, 1, 2, and 3. The sense (level or edge) of each of these signals is programmable. All of these inputs can be driven completely asynchronously.
		State Meaning Asserted—When an external interrupt request signal is asserted, the priority is checked by the IPIC unit, and the interrupt is conditionally passed to the processor. Negated—There is no incoming interrupt from that source.
		Timing Assertion—All of these inputs can be asserted completely asynchronously. Negation—Interrupts programmed as level-sensitive must remain asserted until serviced.
$\overline{\text{MCP_OUT}}$	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 <i>mcp</i> interrupts generated by on-chip sources. See Section 9.3 , “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.

9.5 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 9-3](#) shows the memory map of the IPIC unit. A special set of registers are the QUICC Engine Ports Interrupts registers. These registers have a different base address in the global memory map, and they are detailed in [Table 9-4](#).

Table 9-3. IPIC Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
Integrated Programmable Interrupt Controller—Block Base Address 0x0_0700				
0x00	System global interrupt configuration register (SICFR)	R/W	0x0000_0000	9.5.1/9-7
0x04	System regular interrupt vector register (SIVCR)	R	0x0000_0000	9.5.2/9-8
0x08	System internal interrupt pending register (SIPNR_H)	R	0x0000_0000	9.5.3/9-11
0x0C	System internal interrupt pending register (SIPNR_L)	R	0x0000_0000	9.5.3/9-11
0x10	System internal interrupt group A priority register (SIPRR_A)	R/W	0x0530_9770	9.5.4/9-14
0x14	System internal interrupt group B priority register (SIPRR_B)	R/W	0x0530_9770	9.5.5/9-15
0x18	System internal interrupt group C priority register (SIPRR_C)	R/W	0x0530_9770	9.5.6/9-15
0x1C	System internal interrupt group D priority register (SIPRR_D)	R/W	0x0530_9770	9.5.7/9-16
0x20	System internal interrupt mask register (SIMSR_H)	R/W	0x0000_0000	9.5.8/9-17
0x24	System internal interrupt mask register (SIMSR_L)	R/W	0x0000_0000	9.5.8/9-17
0x28	System internal interrupt control register (SICNR)	R/W	0x0000_0000	9.5.9/9-18
0x2C	System external interrupt pending register (SEPNR)	R/W	Special	9.5.10/9-20
0x30	System mixed interrupt group A priority register (SMPRR_A)	R/W	0x0530_9770	9.5.11/9-21
0x34	System mixed interrupt group B priority register (SMPRR_B)	R/W	0x0530_9770	9.5.12/9-22
0x38	System external interrupt mask register (SEMSR)	R/W	0x0000_0000	9.5.13/9-22
0x3C	System external interrupt control register (SECNR)	R/W	0x0000_0000	9.5.14/9-23
0x40	System error status register (SERSR)	R/W	0x0000_0000	9.5.15/9-25
0x44	System error mask register (SERMR)	R/W		9.5.16/9-26
0x48	System error control register (SERCR)	R/W	0x0000_0000	9.5.17/9-27
0x4C	System external interrupt polarity control register (SEPCR)	R/W	0x0000_0000	9.5.18/9-27
0x4F	Reserved	—	—	—
0x50	System internal interrupt force register (SIFCR_H)	R/W	0x0000_0000	9.5.19/9-28
0x54	System internal interrupt force register (SIFCR_L)	R/W	0x0000_0000	9.5.19/9-28
0x58	System external interrupt force register (SEFCR)	R/W	0x0000_0000	9.5.20/9-30
0x5C	System error force register (SERFR)	R/W	0x0000_0000	9.5.21/9-30

Table 9-3. IPIC Register Address Map (continued)

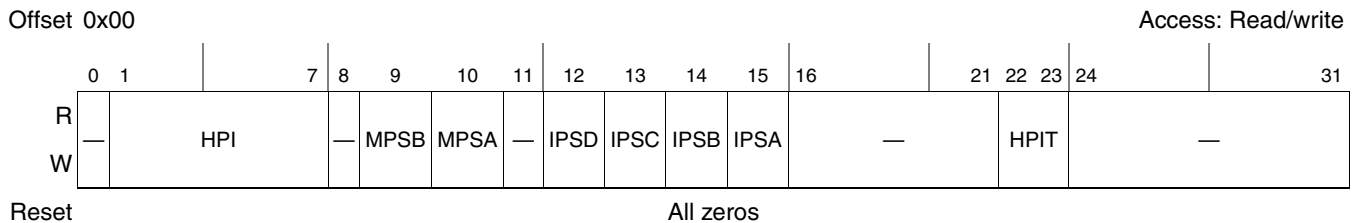
Offset	Register	Access	Reset Value	Section/ Page
0x60	System critical interrupt vector register (SCVCR)	R	0x0000_0000	9.5.22/9-31
0x64	System management interrupt vector register (SMVCR)	R	0x0000_0000	9.5.23/9-31
0x68–0xBF	Reserved	—	—	—

Table 9-4. QUICC Engine Ports Interrupts Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
0x0C	QUICC Engine ports interrupt event register (CEPIER)	w1c	Special	9.5.24/9-32
0x10	QUICC Engine ports interrupt mask register (CEPIMR)	R/W	0x0000_0000	9.5.25/9-33
0x14	QUICC Engine ports interrupt control register (CEPICR)	R/W	0x0000_0000	9.5.26/9-35

9.5.1 System Global Interrupt Configuration Register (SICFR)

SICFR, shown in [Figure 9-2](#), defines the highest priority interrupt and whether interrupts are grouped or spread in the priority table.


Figure 9-2. System Global Interrupt Configuration Register (SICFR)

[Table 9-5](#) defines the bit fields of SICFR.

Table 9-5. 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 9-7) of the single interrupt controller interrupt source that is advanced to the highest priority in the IPIC priority table (see Table 9-38). 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.

Table 9-5. SICFR Field Descriptions (continued)

Bits	Name	Description
10	MPSA	Mixed interrupts priority scheme for group A. Selects the relative MIXA priority scheme. It cannot be changed dynamically. 0 Grouped. The MIXAs are grouped by priority at the top of the table. 1 Spread. The MIXAs are spread by priority in the table.
11	—	Write ignored, read = 0
12	IPSD	Internal interrupts priority scheme for group D. Selects the relative SYSD priority scheme. It cannot be changed dynamically. 0 Grouped. The SYSDs are grouped by priority at the top of the table. 1 Spread. The SYSDs are spread by priority in the table.
13	IPSC	Internal interrupts priority scheme for group C. Selects the relative SYSC priority scheme. It cannot be changed dynamically. 0 Grouped. The SYSCs are grouped by priority at the top of the table. 1 Spread. The SYSCs are spread by priority in the table.
14	IPSB	Internal interrupts priority scheme for group B. Selects the relative SYSB priority scheme. It cannot be changed dynamically. 0 Grouped. The SYSBs are grouped by priority at the top of the table. 1 Spread. The SYSBs are spread by priority in the table.
15	IPSA	Internal interrupts priority scheme for group A. Selects the relative SYSA priority scheme. It cannot be changed dynamically. 0 Grouped. The SYSAs are grouped by priority at the top of the table. 1 Spread. The SYSAs are spread by priority in the table.
16–21	—	Write ignored, read = 0
22–23	HPIT	HPI priority position IPIC output interrupt type. Defines which type of IPIC output interrupt signal (\overline{int} , \overline{cint} , or \overline{smi}) asserts its request to the core in the HPI priority position. These bits cannot be changed dynamically. (If software really wants to change it, it has to make sure the corresponding interrupt source is masked or it won't happen during the change). The definition of HPIT is as follows: 00 \overline{int} request is asserted to the core for HPI. 01 \overline{smi} request is asserted to the core for HPI. 10 \overline{cint} request is asserted to the core for HPI. 11 Reserved.
24–31	—	Write ignored, read = 0

9.5.2 System Global Interrupt Vector Register (SIVCR)

SIVCR, shown in [Figure 9-3](#), contains a 7-bit code ([Table 9-6](#)) representing the regular unmasked interrupt source (\overline{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).

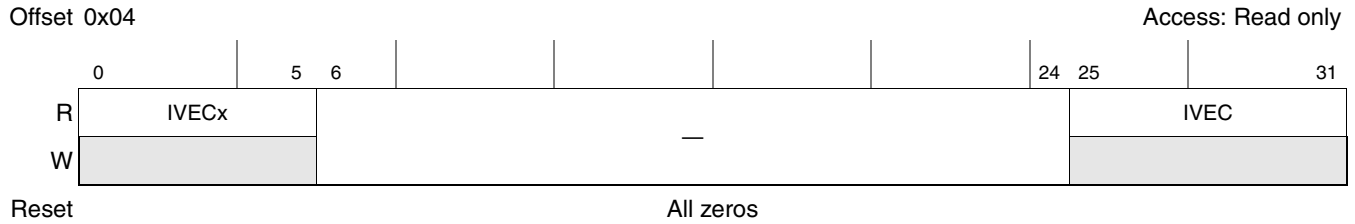


Figure 9-3. System Global Interrupt Vector Register (SIVCR)

Table 9-6 defines the bit fields of SIVCR.

Table 9-6. 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 9-7 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 9-7 for details). The value of SIVCR cannot change while it is being read.

Table 9-7 shows the definition of IVEC.

Table 9-7. IVEC/CVEC/MVEC Field Definition

Interrupt ID Number	Interrupt Meaning	Interrupt Vector
0	Error (no interrupt)	0b000_0000
1–2	Reserved	0b000_0001–0b000_0010
3	DMA Engine 1	0b000_0011
4–8	Reserved	0b000_0100–0b000_1000
9	UARTx	0b000_1001
10	FlexCANx	0b000_1010
11–13	Reserved	0b000_1011–0b000_1101
14	I2C1	0b000_1110
15	I2C2	0b000_1111
16	SPI	0b001_0000
17	IRQ1	0b001_0001
18	IRQ2	0b001_0010
19	IRQ3	0b001_0011
20–31	Reserved	0b001_0100–0b001_1111
32	QUICC Engine High	0b010_0000
33	QUICC Engine Low	0b010_0001
34–37	Reserved	0b010_0010–0b010_1111
38	USB DR	0b010_0110
39–41	Reserved	0b010_0111–0b010_1001
42	eSDHC	0b010_1010
43–47	Reserved	0b010_1011–0b010_1111
48	IRQ0	0b011_0000
49–63	Reserved	0b011_0001–0b011_1111
64	RTC SEC	0b100_0000
65	PIT	0b100_0001
66	PCI	0b100_0010
67	MSIR0	0b100_0011
68	RTC ALR	0b100_0100
69	MU	0b100_0101
70	SBA	0b100_0110
71	IOU/ DMAC	0b100_0111
72	GTM4	0b100_1000
73	GTM8	0b100_1001
74	QUICC Engine Ports	0b100_1010

Table 9-7. IVEC/CVEC/MVEC Field Definition (continued)

Interrupt ID Number	Interrupt Meaning	Interrupt Vector
75	GPIOx	0b100_1011
76	DDR	0b100_1100
77	eLBC	0b100_1101
78	GTM2	0b100_1110
79	GTM6	0b100_1111
80	PMC	0b101_0000
81	MSIR2	0b101_0001
82	MSIR3	0b101_0010
83	Reserved	0b101_0011
84	GTM3	0b101_0100
85	GTM7	0b101_0101
86	MSIR4	0b101_0110
87	MSIR5	0b101_0111
88	MSIR6	0b101_1000
89	MSIR7	0b101_1001
90	GTM1	0b101_1010
91	GTM5	0b101_1011
92–93	Reserved	0b101_1100–0b101_1101
94	DMA Engine 1 ERR	0b101_1110
95	DPTC	0b101_1111
96–127	Reserved	0b110_0000–0b111_1111

9.5.3 System Internal Interrupt Pending Registers (SIPNR_H and SIPNR_L)

Each bit in SIPNR_H and SIPNR_L, shown in [Figure 9-4](#) and [Figure 9-5](#), may be assigned an internal interrupt source (implemented bits are listed in [Table 9-8](#).) 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.

Integrated Programmable Interrupt Controller (IPIC)



Figure 9-4. System Internal Interrupt Pending Register (SIPNR_H)

Table 9-8 lists implemented SIPNR_H fields. Note that these field descriptions are also valid for SIFCR_H and SIMSR_H.

Table 9-8. SIPNR_H/SIFCR_H/SIMSR_H Bit Assignments

Bits	Field
0	QUICC Engine High
1	QUICC Engine Low
2–5	Reserved
6	USBDR
7–9	Reserved
10	eSDHC
11–17	Reserved
18	DMA Engine 1
19	MSIR1
20–23	Reserved
24	UARTx
25	FlexCANx
26–28	Reserved
29	I2C1
30	I2C2
31	SPI

Table 9-9 defines the bit fields of SIPNR_H.

Table 9-9. SIPNR_H Field Descriptions

Bits	Name	Description
0–31	INT _n	Each implemented bit (listed in Table 9-8) 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 9-4](#).

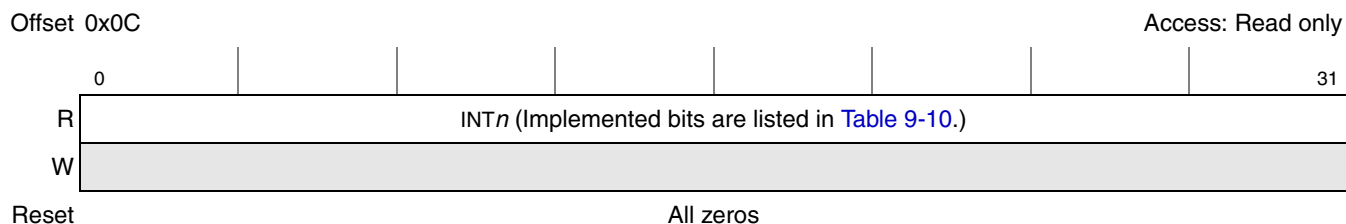


Figure 9-5. System Internal Interrupt Pending Register (SIPNR_L)

[Table 9-10](#) lists implemented SIPNR_L fields. Note that these field assignments are also valid for SIFCR_L and SIMSR_L.

Table 9-10. SIPNR_L/SIFCR_L/SIMSR_L Bit Assignments

Bits	Field
0	RTC SEC
1	PIT
2	PCI
3	MSIR0
4	RTC ALR
5	MU
6	SBA
7	IOU/ DMAC
8	GTM4
9	GTM8
10	QUICC Engine Port
11	GPIO n
12	DDR
13	eLBC
14	GTM2
15	GTM6
16	PMC
17	MSIR2
18	MSIR3
19	—
20	GTM3
21	GTM7
22	MSIR4
23	MSIR5
24	MSIR6
25	MSIR7
26	GTM1
27	GTM5

Table 9-10. SIPNR_L/SIFCR_L/SIMSR_L Bit Assignments (continued)

Bits	Field
28–29	—
30	DMA Engine 1 ERR
31	—

Table 9-11 defines the bit fields of SIPNR_L.

Table 9-11. SIPNR_L Field Descriptions

Bits	Name	Description
0–31	INT n	Each implemented bit (listed in Table 9-10) 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.

9.5.4 System Internal Interrupt Group A Priority Register (SIPRR_A)

The system internal interrupt group A priority register (SIPRR_A), shown in Figure 9-6, defines the priority between USBDR, QUICC Engine High and QUICC Engine Low internal interrupt signals.

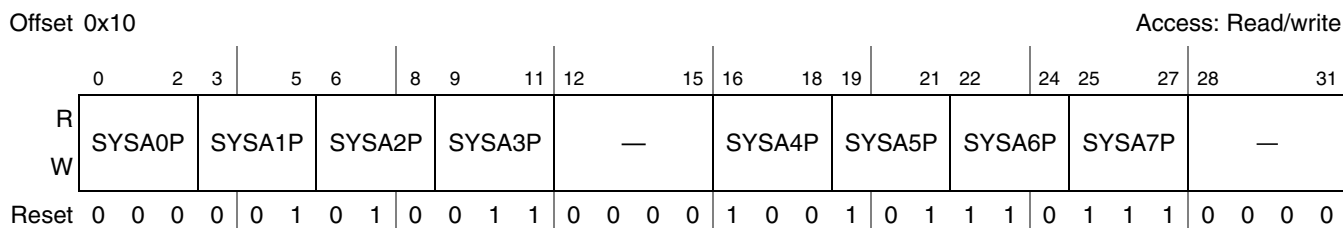


Figure 9-6. System Internal Interrupt Group A Priority Register (SIPRR_A)

Table 9-12 defines the bit fields of SIPRR_A.

Table 9-12. 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 QUICC Engine High asserts its request in the SYSA0 position. 001 QUICC Engine Low asserts its request in the SYSA0 position. 010–101 Reserved 110 USB DR 111 Reserved
3–11, 16–27	SYSA1P–SYSA7P	Same as SYSA0P, but for SYSA1P–SYSA7P.
12–15, 28–31	—	Write ignored, read = 0

9.5.5 System Internal Interrupt Group B Priority Register (SIPRR_B)

The system internal interrupt group B priority register (SIPRR_B), shown in [Figure 9-7](#), defines the priority between internal interrupt signals.

For more information, see [Section 9.6.3, “Internal Interrupts Group Relative Priority.”](#)

[Table 9-13](#) defines the bit fields of SIPRR_B.

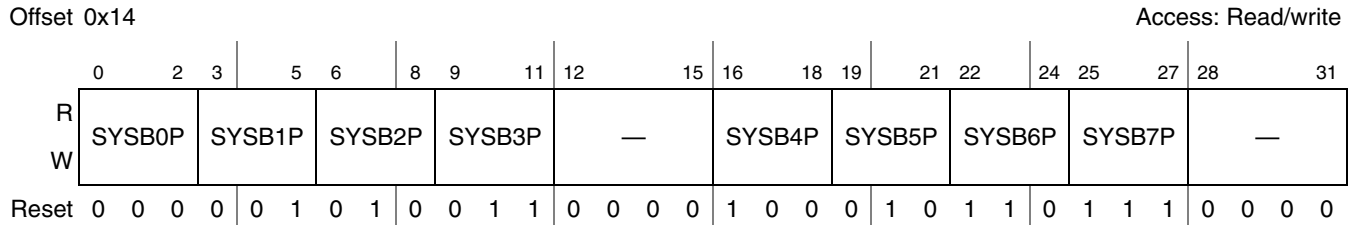


Figure 9-7. System Internal Interrupt Group B Priority Register (SIPRR_B)

Table 9-13. SIPRR_B Field Descriptions

Bits	Name	Description
0–2	SYSB0P	SYSB0 Priority order. Defines which interrupt source asserts its request in the SYSB0 priority position. The user should not program the same code to more than one priority position (0–7). These bits can be changed dynamically. The definition of SYSB0P is shown as follows: 000–001 Reserved 010 eSDHC asserts its request in the SYSB0 position. 011–111 Reserved
3–11, 16–27	SYSB1P– SYSB7P	Same as SYSB0P, but for SYSB1P–SYSB7P.
12–15, 28–31	—	Write ignored, read = 0

9.5.6 System Internal Interrupt Group C Priority Register (SIPRR_C)

The system internal interrupt group C priority register (SIPRR_C), shown in [Figure 9-8](#), defines the priority between internal interrupt signals.

For more information about interrupt priorities, see [Section 9.6.3, “Internal Interrupts Group Relative Priority.”](#)

Table 9-14 defines the bit fields of SIPRR_C.

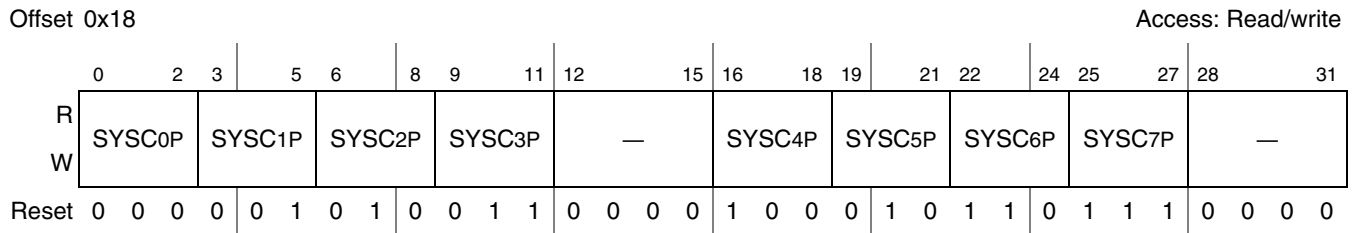


Figure 9-8. System Internal Interrupt Group C Priority Register (SIPRR_C)

Table 9-14. SIPRR_C Field Descriptions

Bits	Name	Description
0–2	SYSC0P	SYSC0 priority order. Defines which interrupt source asserts its request in the SYSC0 priority position. The user should not program the same code to more than one priority position (0–7). These bits can be changed dynamically. The definition of SYSC0P is shown as follows: 000–001 Reserved 010 DMA Engine 1 011–111 Reserved
3–11, 16–27	SYSC1P– SYSC7P	Same as SYSC0P, but for SYSC1P–SYSC7P.
12–15, 28–31	—	Write ignored, read = 0

9.5.7 System Internal Interrupt Group D Priority Register (SIPRR_D)

SIPRR_D, shown in Figure 9-9, defines the priority among the interrupt sources listed in Table 9-15.

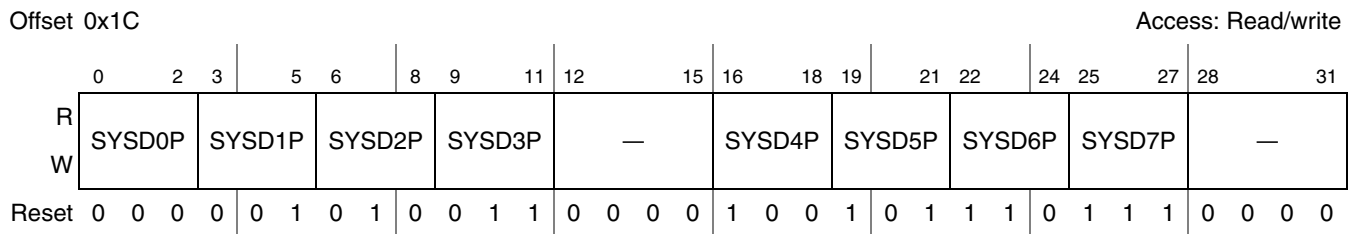


Figure 9-9. System Internal Interrupt Group D Priority Register (SIPRR_D)

Table 9-15 defines the bit fields of SIPRR_D.

Table 9-15. 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 UARTx asserts its request in the SYSD0 position. 001 FlexCANx asserts its request in the SYSD0 position. 010–100 Reserved 101 I2C1 asserts its request in the SYSD0 position. 110 I2C2 asserts its request in the SYSD0 position.Reserved 111 SPI asserts its request in the SYSD0 position.Reserved
3–11, 16–27	SYSD1P– SYSD7P	Same as SYSD0P, but for SYSD1P–SYSD7P.
12–15, 28–31	—	Write ignored, read = 0

9.5.8 System Internal Interrupt Mask Register (SIMSR_H and SIMSR_L)

Each implemented bit in SIMSR_H and SIMSR_L, shown in [Figure 9-10](#) and [Figure 9-11](#), corresponds to an internal interrupt source. The user masks an interrupt by clearing the corresponding SIMSR bit. When an interrupt request occurs, the corresponding SIPNR bit is set, regardless of the SIMSR bit. However, if the corresponding SIMSR bit is cleared, no interrupt request is passed to the core.

When 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.

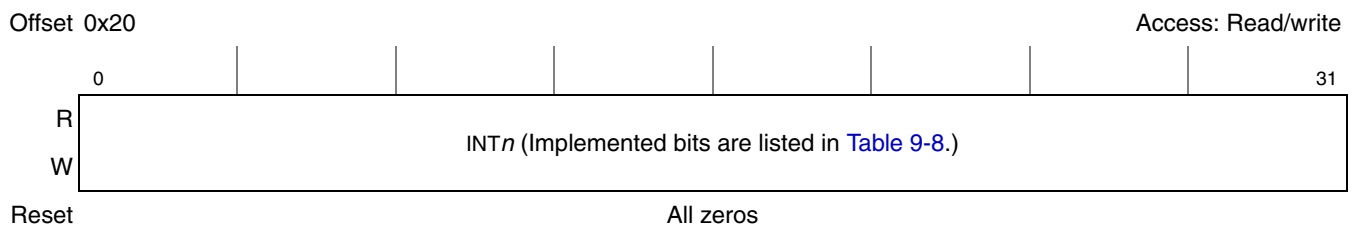


Figure 9-10. System Internal Interrupt Mask Register (SIMSR_H)

Table 9-16 defines the bit fields of SIMSR_H.

Table 9-16. SIMSR_H Field Descriptions

Bits	Name	Description
0–31	INT _n	<p>Each implemented bit (listed in Table 9-8) corresponds to an external interrupt source. The user masks an interrupt by clearing the corresponding SIMSR bit. An interrupt is unmasked (enable) by setting the corresponding SIMSR bit. The SIMSR can be read by the user at any time.</p> <p>Note:</p> <ul style="list-style-type: none"> • 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 <code>rfi</code> instruction. The error vector cannot be masked. <p>Unimplemented bits, shown as reserved in Table 9-8, are ignored on writes; read = 0.</p>

Figure 9-11 shows SIMSR_L.

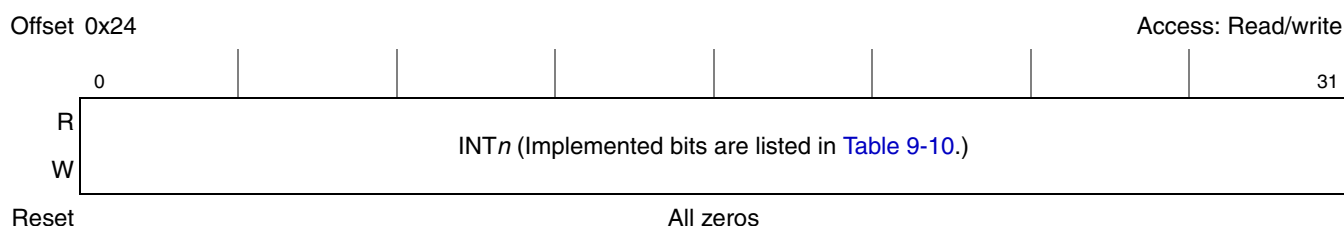


Figure 9-11. System Internal Interrupt Mask Register (SIMSR_L)

Table 9-17 defines the bit fields of SIMSR_L.

Table 9-17. SIMSR_L Field Descriptions

Bits	Name	Description
0–31	INT _n	<p>Each implemented bit (listed in Table 9-10) corresponds to an external interrupt source. The user masks an interrupt by clearing the corresponding SIMSR bit. An interrupt is unmasked (enabled) by setting the corresponding SIMSR bit. The SIMSR can be read by the user at any time.</p> <p>Note:</p> <ul style="list-style-type: none"> • 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 <p>Unimplemented bits, shown as reserved in Figure 9-11, are ignored on writes; read = 0.</p>

9.5.9 System Internal Interrupt Control Register (SICNR)

SICNR, shown in Figure 9-12, defines the IPIC output interrupt type ($\overline{\text{int}}$, $\overline{\text{cint}}$, or $\overline{\text{smi}}$) in the SYSA0–SYSA1, SYSB0–SYSB1, SYSC0–SYSC1, 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{int} output interrupt type (should not use \overline{cint} or \overline{smi} output interrupt types) to read an updated SIVCR.

Offset 0x28

Access: Read write

	0	1	2	3	4	7	8	9	10	11	12	15	16	17	18	19	20	23	24	25	26	27	28	31		
R	SYSD0T	SYSD1T	—	—	—	—	SYSC0T	SYSC1T	—	—	—	—	SYSB0T	SYSB1T	—	—	—	—	SYSA0T	SYSA1T	—	—	—	—		
W																										
Reset	All zeros																									

Figure 9-12. System Internal Interrupt Control Register (SICNR)

Table 9-18 defines the bit fields of SICNR.

Table 9-18. 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 (\overline{int} , \overline{cint} , or \overline{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 \overline{int} request is asserted to the core for SYSD0. 01 \overline{smi} request is asserted to the core for SYSD0. 10 \overline{cint} request is asserted to the core for SYSD0. 11 Reserved
2–3	SYSD1T	Same as SYSD0T, but for SYSD1T.
4–7	—	Write ignored, read = 0
8–9	SYSC0T	SYSC0 priority position IPIC output interrupt Type. Defines which type of the IPIC output interrupt signal (\overline{int} , \overline{cint} , or \overline{smi}) asserts its request to the core in the SYSC0 priority position. These bits can not be changed dynamically. (If s/w really wants to change it, it has to make sure the corresponding interrupt source is masked or it won't happen during the change). The definition of SYSC0T is as follows: 00 \overline{int} request is asserted to the core for SYSC0. 01 \overline{smi} request is asserted to the core for SYSC0. 10 \overline{cint} request is asserted to the core for SYSC0. 11 Reserved
10–11	SYSC1T	Same as SYSC0T, but for SYSC1T.
12–15	—	Write ignored, read = 0
16–17	SYSB0T	SYSB0 priority position IPIC output interrupt Type. Defines which type of the IPIC output interrupt signal (\overline{int} , \overline{cint} , or \overline{smi}) asserts its request to the core in the SYSB0 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 SYSB0T is as follows: 00 \overline{int} request is asserted to the core for SYSB0. 01 \overline{smi} request is asserted to the core for SYSB0. 10 \overline{cint} request is asserted to the core for SYSB0. 11 Reserved
18–19	SYSB1T	Same as SYSB0T, but for SYSB1T.
20–23	—	Write ignored, read = 0

Table 9-18. SICNR Field Descriptions (continued)

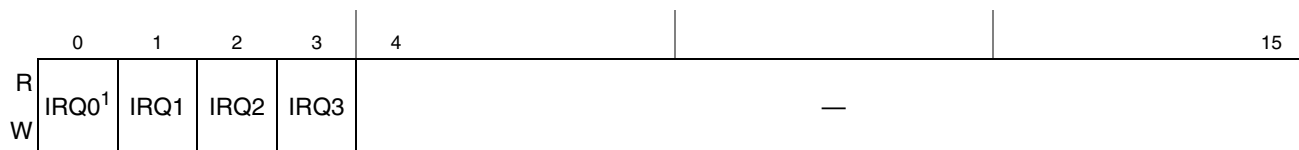
Bits	Name	Description
24–25	SYSA0T	SYSA0 priority position IPIC output interrupt type. Defines which type of the IPIC output interrupt signal (\overline{int} , \overline{smi} , or \overline{cini}) asserts its request to the core in the SYSA0 priority position. These bits can not be changed dynamically. (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 \overline{int} request is asserted to the core for SYSA0. 01 \overline{smi} request is asserted to the core for SYSA0. 10 \overline{cini} 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

9.5.10 System External Interrupt Pending Register (SEPNR)

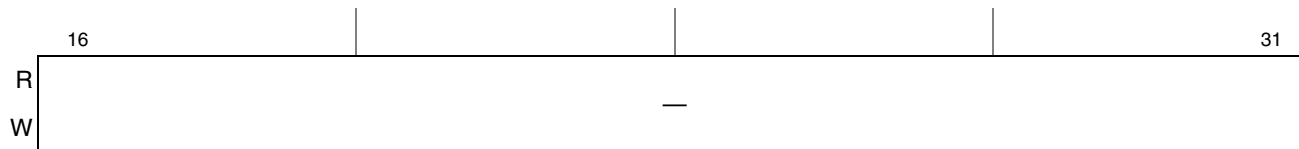
Each bit in the SEPNR, shown in Figure 9-13, corresponds to an external interrupt source. When an interrupt is received, the interrupt controller sets the corresponding SEPNR bit.

Offset 0x2C

Access: Read/write



Reset The reset values of implemented bits reflect the values of the external IRQ signals. Reserved bits are zeros. ²



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 9-13. System External Interrupt Pending Register (SEPNR)

Table 9-19 defines the bit fields of SEPNR.

Table 9-19. SEPNR Field Descriptions

Bits	Name	Description
0, 1, 2, 3	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 $\overline{\text{IRQ}}_n$ to negate which automatically clears the bit in SEPNR. For edge-triggered cases, the software needs to clear the corresponding bit in SEPNR. SEPNR bits are cleared by writing ones to them. Because the user can only clear bits in this register, writing zeros to this register has no effect. Note that the SEPNR bit positions are not changed according to their relative priority.
4–31	—	Write ignored, read = 0

9.5.11 System Mixed Interrupt Group A Priority Register (SMPRR_A)

The SMPRR_A, shown in Figure 9-14, defines the priority among the sources listed in Table 9-20.

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	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 9-14. System Mixed Interrupt Group A Priority Register (SMPRR_A)

Table 9-20 defines the bit fields of SMPRR_A.

Table 9-20. SMPRR_A Field Descriptions

Bits	Name	Description
0–2	MIXA0P	MIXA0 priority order. Defines which interrupt source asserts its request in the MIXA0 priority position. The user must not program the same code to more than one priority position (0–7). These bits can be changed dynamically. The definition of MIXA0P is as follows: 000 RTC SEC asserts its request to the MIXA0 position. 001 PIT asserts its request to the MIXA0 position. 010 Reserved 011 Reserved. 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

9.5.12 System Mixed Interrupt Group B Priority Register (SMPRR_B)

SMPRR_B, shown in [Figure 9-15](#), defines the priority among the sources listed in [Table 9-21](#).

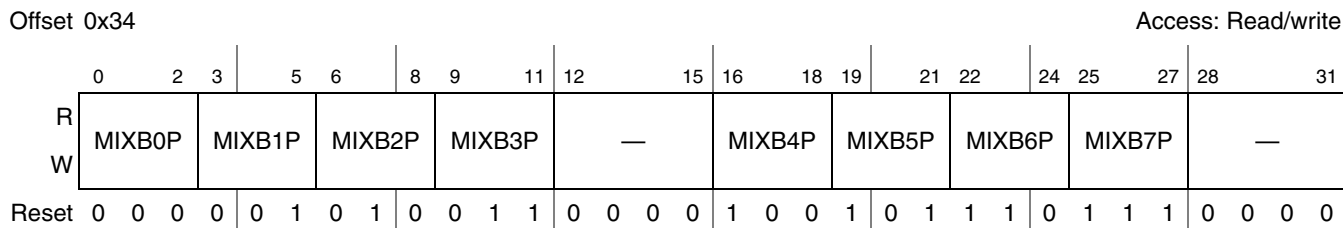


Figure 9-15. System Mixed Interrupt Group B Priority Register (SMPRR_B)

[Table 9-21](#) defines the bit fields of SMPRR_B.

Table 9-21. SMPRR_B Field Descriptions

Bits	Name	Description
0–2 3–11, 16–27	MIXBnP	MIXBn priority order. Defines which interrupt source asserts its request in the MIXBn priority position. The user must not program the same code to more than one priority position (0–7). These bits can be changed dynamically. The definition of MIXBnP is as follows: 000 RTC ALR asserts its request to the MIXBn position. 001 Reserved. 010 SBA asserts its request to the MIXBn position. 011 DMA Engine 2 asserts its request to the MIXBn position. 100–111 Reserved
12–15, 28–31	—	Write ignored, read = 0

9.5.13 System External Interrupt Mask Register (SEMSR)

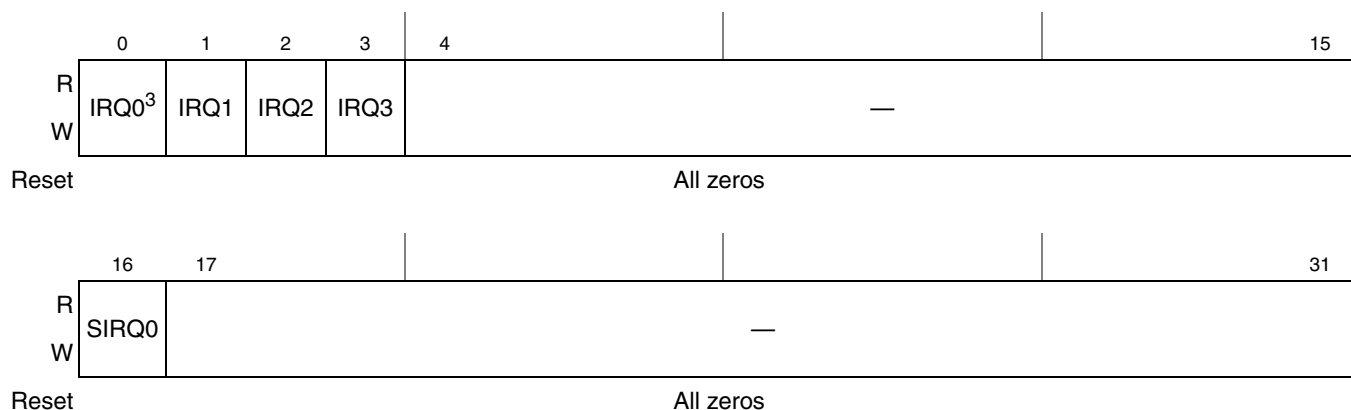
Each bit in the system external interrupt mask register (SEMSR), shown in [Figure 9-13](#), corresponds to an external interrupt source. The user masks an interrupt by clearing the corresponding SEMSR bit. An interrupt is unmasked (enabled) by setting the corresponding SEMSR bit.

When an external interrupt request occurs, the corresponding 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



³ This bit is valid only if the IRQ0 signal is configured as an external maskable interrupt (SEMSR[SIRQ0] = 0)

Figure 9-16. System External Interrupt Mask Register (SEMSR)

Table 9-22 defines the bit fields of SEMSR.

Table 9-22. SEMSR Field Descriptions

Bits	Name	Description
0, 1, 2, 3	—	Each bit corresponds to an external interrupt source. The user masks an interrupt by clearing the SEMSR bit. An interrupt can be enabled by setting the corresponding SEMSR bit. SEMSR can be read by the user at any time. Note: <ul style="list-style-type: none"> 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 SEP NR bit causes an interrupt request to the core, the error vector is issued (if no other interrupts pending). Thus, the user must always include an error vector routine, even if it contains only an <i>rfi</i> instruction. The error vector cannot be masked.
4–15	—	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

9.5.14 System External Interrupt Control Register (SECNR)

SECNR, shown in Figure 9-17, defines the edge detect mode for external \overline{IRQn} interrupt signals and determines whether the corresponding \overline{IRQn} 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 (*int*, *cint*, or *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 *cint* or *smi* output interrupt types) in order to read an updated SIVCR.

Offset 0x3C

Access: Read/write

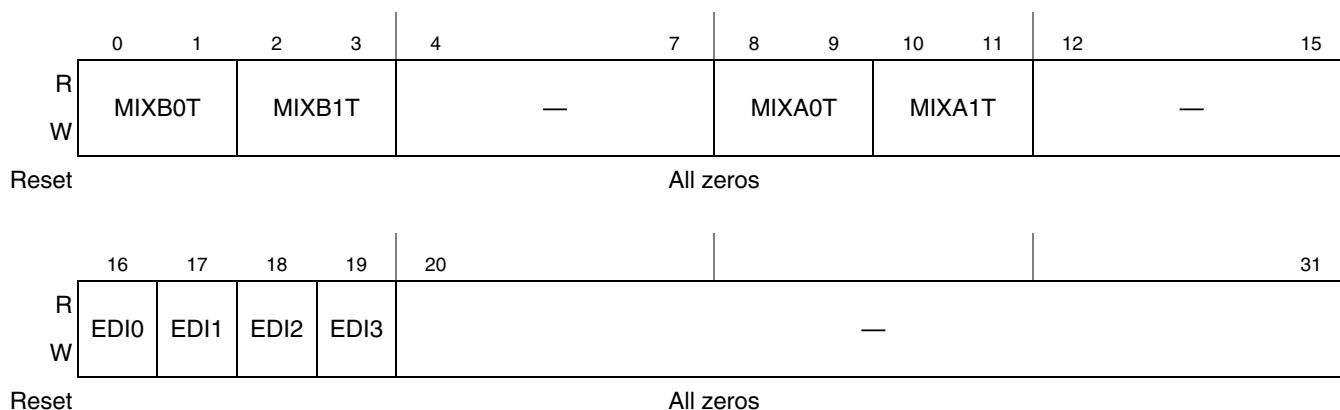


Figure 9-17. System External Interrupt Control Register (SECNR)

Table 9-23 defines the bit fields of SECNR.

Table 9-23. SECNR Field Descriptions

Bits	Name	Description
0–1	MIXB0T	MIXB0 priority position IPIC output interrupt type. Defines which type of the IPIC output interrupt signal (\overline{int} , \overline{cint} , or \overline{smi}) asserts its request to the core in the MIXB0 priority position. These bits can be changed dynamically. The definition of MIXB0T is as follows: 00 \overline{int} request is asserted to the core for MIXB0. 01 \overline{smi} request is asserted to the core for MIXB0. 10 \overline{cint} request is asserted to the core for MIXB0. 11 Reserved
2–3	MIXB1T	Same as MIXB0T, but for MIXB1T.
4–7	—	Write ignored, read = 0
8–9	MIXA0T	MIXA0 priority position IPIC output interrupt Type. Defines which type of the IPIC output interrupt signal (\overline{int} , \overline{cint} , or \overline{smi}) asserts its request to the core in the MIXA0 priority position. These bits can be changed dynamically. The definition of MIXA0T is as follows: 00 \overline{int} request is asserted to the core for MIXA0. 01 \overline{smi} request is asserted to the core for MIXA0. 10 \overline{cint} request is asserted to the core for MIXA0. 11 Reserved
10–11	MIXA1T	Same as MIXA0T, but for MIXA1T.
12–15	—	Write ignored, read = 0
16–19	EDIx	Each bit defines the edge detect mode for the external \overline{IRQn} interrupt signals, determines whether the corresponding \overline{IRQn} signal asserts an interrupt request upon either a high-to-low (high assertion for active high polarity) change or low assertion (high assertion for active high polarity) on the pin. The corresponding \overline{IRQn} signal asserts an interrupt request as follows: 0 Low assertion (high assertion for active high polarity) on \overline{IRQn} generates an interrupt request (level sensitive). 1 High-to-low (low to high assertion for active high polarity) change on \overline{IRQn} generates an interrupt request (edge sensitive).
20–31	—	Write ignored, read = 0

9.5.15 System Error Status Register (SERSR)

The bits in the SERSR, shown in [Figure 9-18](#), correspond to the external and internal non-maskable error source machine check (mcp) conditions listed in [Table 9-24](#). When an error interrupt signal is received, the interrupt controller sets the corresponding SERSR bit.



Figure 9-18. System Error Status Register (SERSR)

[Table 9-24](#) lists the implemented SERSR bits. Note that these field assignments are valid for SERMR and SERFR.

Table 9-24. SERSR/SERMR/SERFR Bit Assignments

Bits	Field
0	IRQ0 ¹
1	WDT
2	SBA
3	CIEE
4	CMEE
5	—
6	—
7	—
8–31	—

¹ This bit is valid only if the IRQ0 signal is configured as an external MCP interrupt (SEMSR[SIRQ0] = 1)

[Table 9-25](#) defines the bit fields of SERSR.

Table 9-25. SERSR Field Descriptions

Bits	Name	Description
0–31	INT n	Each implemented bit in the SERSR, listed in Table 9-24 , 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 9-24), writes are ignored, read = 0

9.5.16 System Error Mask Register (SERMR)

Each implemented bit in SERMR, shown in [Figure 9-19](#), corresponds to an external and an internal \overline{mcp} source (MCP). The user masks an MCP by clearing and enables an interrupt by setting the corresponding SERMR bit. When a masked MCP occurs, the corresponding SERSR bit is set, regardless of the setting of the corresponding SERMR bit although no MCP request is passed to the core in this case. The SERMR can be read by the user at any time.

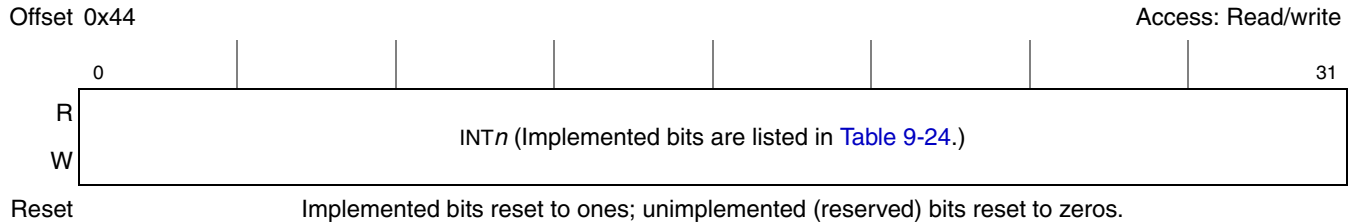


Figure 9-19. System Error Mask Register (SERMR)

[Table 9-26](#) defines the bit fields of SERMR.

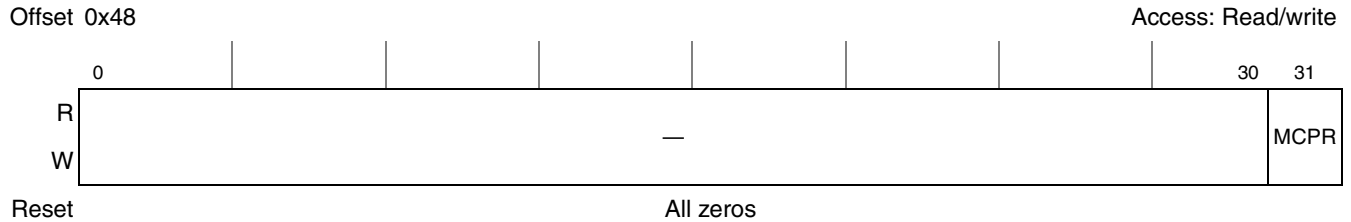
Table 9-26. SERMR Field Descriptions

Bits	Name	Description
0–31	INT n	Each implemented SERMR bit, listed in Table 9-24 , 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

9.5.17 System Error Control Register (SERCR)

SERCR, shown in [Figure](#) , defines the control bits that route MCP requests in core disable mode to $\overline{\text{MCP_OUT}}$.

[Table 9-27](#) defines the bit fields of SERCR.



System Error Control Register (SERCR)

Table 9-27. SERCR Field Descriptions

Bits	Name	Description
0–30	—	Write ignored, read = 0
31	MCPR	MCP route. Route MCP request to $\overline{\text{MCP_OUT}}$. 0 MCP is not available at $\overline{\text{MCP_OUT}}$. 1 MCP routed to $\overline{\text{MCP_OUT}}$.

9.5.18 System External interrupt Polarity Control Register (SEPCR)

SEPCR, shown in [Figure 9-20](#), defines the polarity for each one of the external $\overline{\text{IRQ}}_n$ interrupt signals and determines whether the corresponding $\overline{\text{IRQ}}_n$ signal is treated as active low or active high signal. The active low signals will assert an interrupt request upon either a high-to-low change or assertion (low state) on the pin. The active high signals will assert an interrupt request upon either a low-to-high change or assertion (high state) on the pin. See [Section 9.5.14](#), “System External Interrupt Control Register (SECNR),” on [page 9-23](#) for more details.

NOTE

Note that the $\overline{\text{IRQ}}_n$ signals are overbarred although the SEPCR could be programmed to accept active high signals. The overbar should be ignored in this case.

Table 9-28 defines the bit fields of SEPCR.

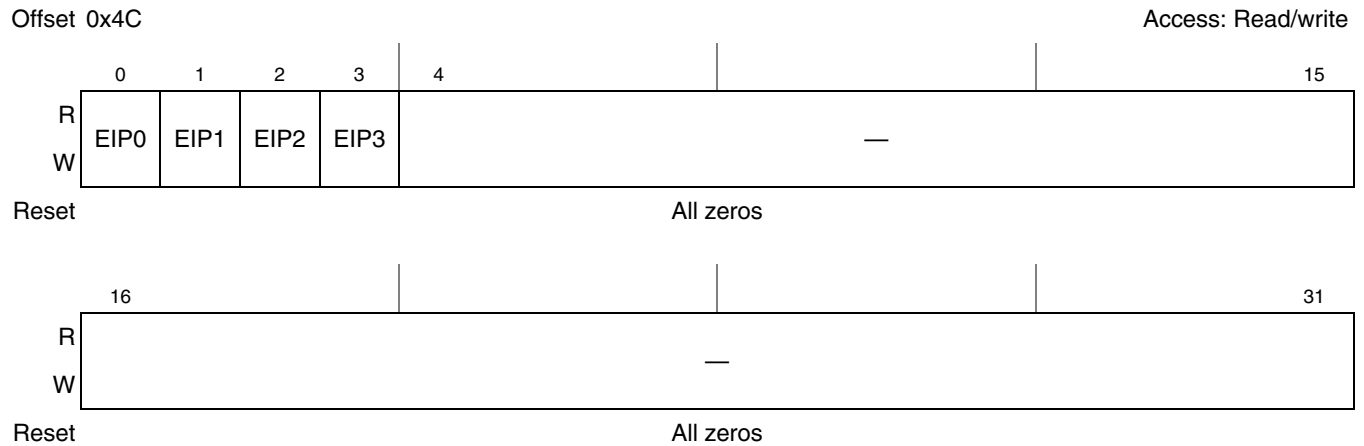


Figure 9-20. System External Interrupt Polarity Control Register (SEPCR)

Table 9-28. SEPCR Field Descriptions

Bits	Name	Description
0–3	EIPx	Each bit defines the active state for the external $\overline{IRQ}n$ interrupt signals. 0 Active Low. 1 Active High.
4–31	—	Write ignored, read = 0

9.5.19 System Internal Interrupt Force Registers (SIFCR_H and SIFCR_L)

Each implemented bit SIFCR_H and SIFCR_L, shown in Figure 9-21 and Figure 9-22, 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 9-21. System Internal Interrupt Force Register (SIFCR_H)

Table 9-29 defines the bit fields of SIFCR_H.

Table 9-29. SIFCR_H Field Descriptions

Bits	Name	Description
0–31	INT n	Each implemented bit, listed in Table 9-8, corresponds to an internal 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 9-22.



Figure 9-22. System Internal Interrupt Force Register (SIFCR_L)

Table 9-30 defines the bit fields of SIFCR_L.

Table 9-30. SIFCR_L Field Descriptions

Bits	Name	Description
0–31	INT n	Each implemented bit, listed in Table 9-10, corresponds to an internal interrupt source. The user forces an interrupt by setting the corresponding SIFCR x bit. SIFCR x bit positions are not changed according to their relative priority. Writes to unimplemented (reserved) bits are ignored; read = 0

9.5.20 System External Interrupt Force Register (SEFCR)

Each implemented bit in SEFCR, shown in [Figure 9-23](#), 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 9-23. System External Interrupt Force Register (SEFCR)

[Table 9-31](#) defines the bit fields of SEFCR.

Table 9-31. SEFCR Field Descriptions

Bits	Name	Description
0, 1, 2, 3	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.
-31	—	Write ignored, read = 0

9.5.21 System Error Force Register (SERFR)

Each bit in the system error force register (SERFR), shown in [Figure 9-24](#), 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 9-24. System Error Status Register (SERFR)

Table 9-32 defines the bit fields of SERFR.

Table 9-32. SERFR Field Descriptions

Bits	Name	Description
0–31	INT _n	Each implemented bit, listed in Table 9-24, 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

9.5.22 System Critical Interrupt Vector Register (SCVCR)

SCVCR, shown in Figure 9-25, contains a 7-bit code (Table 9-33) representing the unmasked critical interrupt (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).

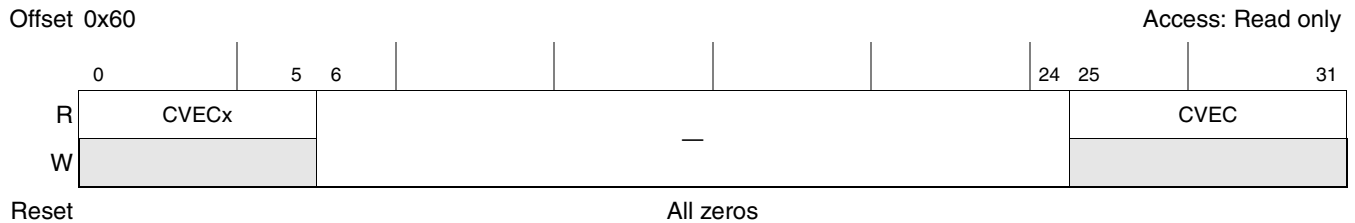


Figure 9-25. System Critical Interrupt Vector Register (SCVCR)

Table 9-33 defines SCVCR bit fields.

Table 9-33. 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 9-7 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 9-7 for details). The value of SCVEC cannot change while it is being read.

9.5.23 System Management Interrupt Vector Register (SMVCR)

SMVCR, shown in Figure 9-26, contains a 7-bit code (Table 9-34) representing the unmasked system management interrupt (SMI) source of the highest priority level.

Note that in core disabled mode the user should use SIVCR only read an updated interrupt vector (SMVCR should not be used).

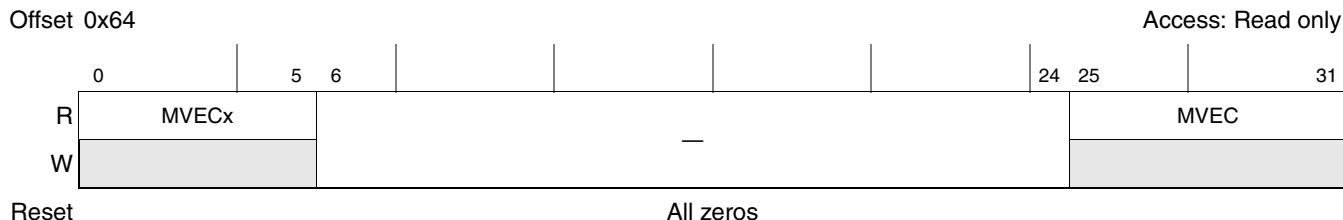


Figure 9-26. System Management Interrupt Vector Register (SMVCR)

Table 9-34 defines the bit fields of SMVCR.

Table 9-34. 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 9-7 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 9-7 for details). The value of SMVEC cannot change while it is being read.

9.5.24 QUICC Engine Ports Interrupt Event Register (CEPIER)

The QUICC Engine ports interrupt event register (CEPIER), shown in Figure 9-27, carries information on the QUICC Engine ports events that caused an interrupt. Each bit in CEPIER, corresponds to one QUICC Engine port, which may be the source of an interrupt. CEPIER bits are cleared by writing ones. However, writing zero has no effect.

Table 9-35 defines the bit fields of CEPIER.

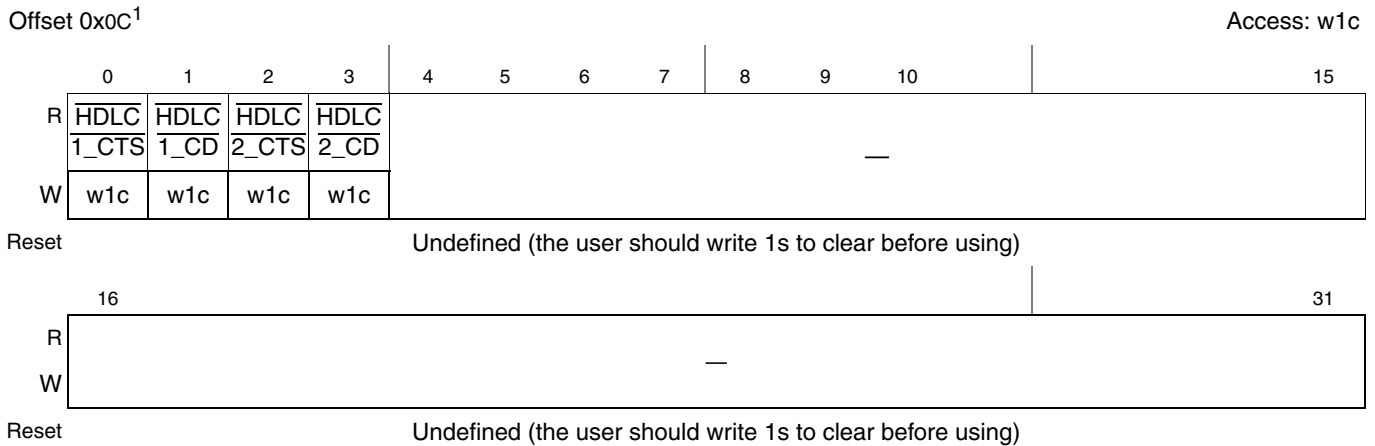


Figure 9-27. QUICC Engine Ports Interrupt Event Register (CEPIER)

¹ Note that the base address for CEPIER is not the same as the base address of other registers in the IPIC. Instead, this register uses a base address which is specific to QUICC engine ports interrupts registers. See [Chapter 2, “Memory Map.”](#)

Table 9-35. CEPIER Bit Settings

Bits	Name	Description
0–3	HDLCx	Interrupt is asserted to indicate change in level of the HDLC CTS/CD pins. Refer to CEPICR for more details.
4–31	—	Reserved. Should be cleared.

9.5.25 QUICC Engine Ports Interrupt Mask Register (CEPIMR)

The QUICC Engine ports interrupt mask register (CEPIMR), shown in [Figure 9-28](#), defines the interrupt masking for the individual QUICC Engine ports lines. When a masked interrupt request occurs, the corresponding CEPIER bit is set, regardless of the CEPIMR state. When one or more non-masked interrupt events occur, the ‘QE Ports’ internal event is generated. The ‘QE Ports’ internal event is one source in the SIPNR register (See [Section 9.5.3, “System Internal Interrupt Pending Registers \(SIPNR_H and SIPNR_L\).”](#))

Table 9-36 defines the bit fields of CEPIMR.

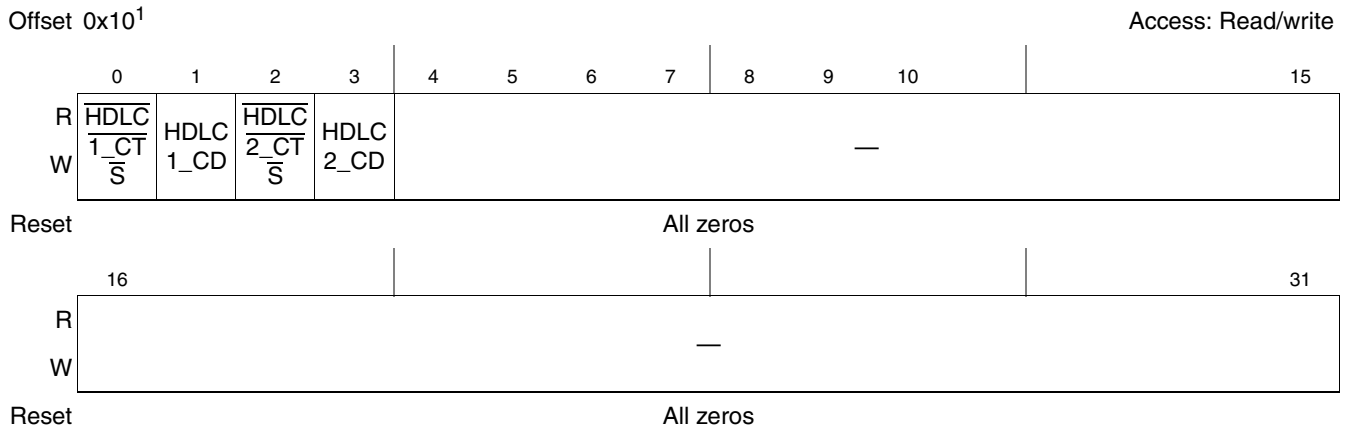


Figure 9-28. QUICC Engine Ports Interrupt Mask Register (CEPIMR)

¹ Note that the base address for CEPIER is not the same as the base address of other registers in the IPIC. Instead, this register uses a base address which is specific to QUICC engine ports interrupts registers. See [Chapter 2, “Memory Map.”](#)

Table 9-36. CEPIMR Bit Settings

Bits	Name	Description
0–3	HDLC x	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).
4–31	—	Reserved. Should be cleared.

9.5.26 QUICC Engine Ports Interrupt Control Register (CEPICR)

The QUICC Engine ports interrupt control register (CEPICR), shown in Figure 9-29, determines whether the corresponding QUICC Engine port line asserts an interrupt request upon either a high-to-low change or any change on the state of the signal.

Table 9-37 defines the bit fields of CEPICR.



Figure 9-29. QUICC Engine Ports Interrupt Control Register (CEPICR)

¹ Note that the base address for CEPIER is not the same as the base address of other registers in the IPIC. Instead, this register uses a base address which is specific to QUICC engine ports interrupts registers. See Chapter 2, “Memory Map.”

Table 9-37. CEPICR Bit Settings

Bits	Name	Description
0–3	HDLC x	Edge detection mode. The corresponding QUICC Engine port line asserts an interrupt request according to the following: 0 Any change on the state of the QUICC Engine port generates an interrupt request. 1 High-to-low change on the QUICC Engine port generates an interrupt request.
4–31	—	Reserved. Should be cleared.

9.6 Functional Description

The following sections describe the types of interrupts, interrupt configurations, and their priorities.

9.6.1 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 *int* signal is the main interrupt output from the IPIC to the processor core and causes the external interrupt exception. The *crit* signal is the critical interrupt output from the IPIC to the processor core and causes the critical external interrupt exception. The *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.

9.6.2 Interrupt Configuration

Figure 9-30 shows the interrupt configuration.

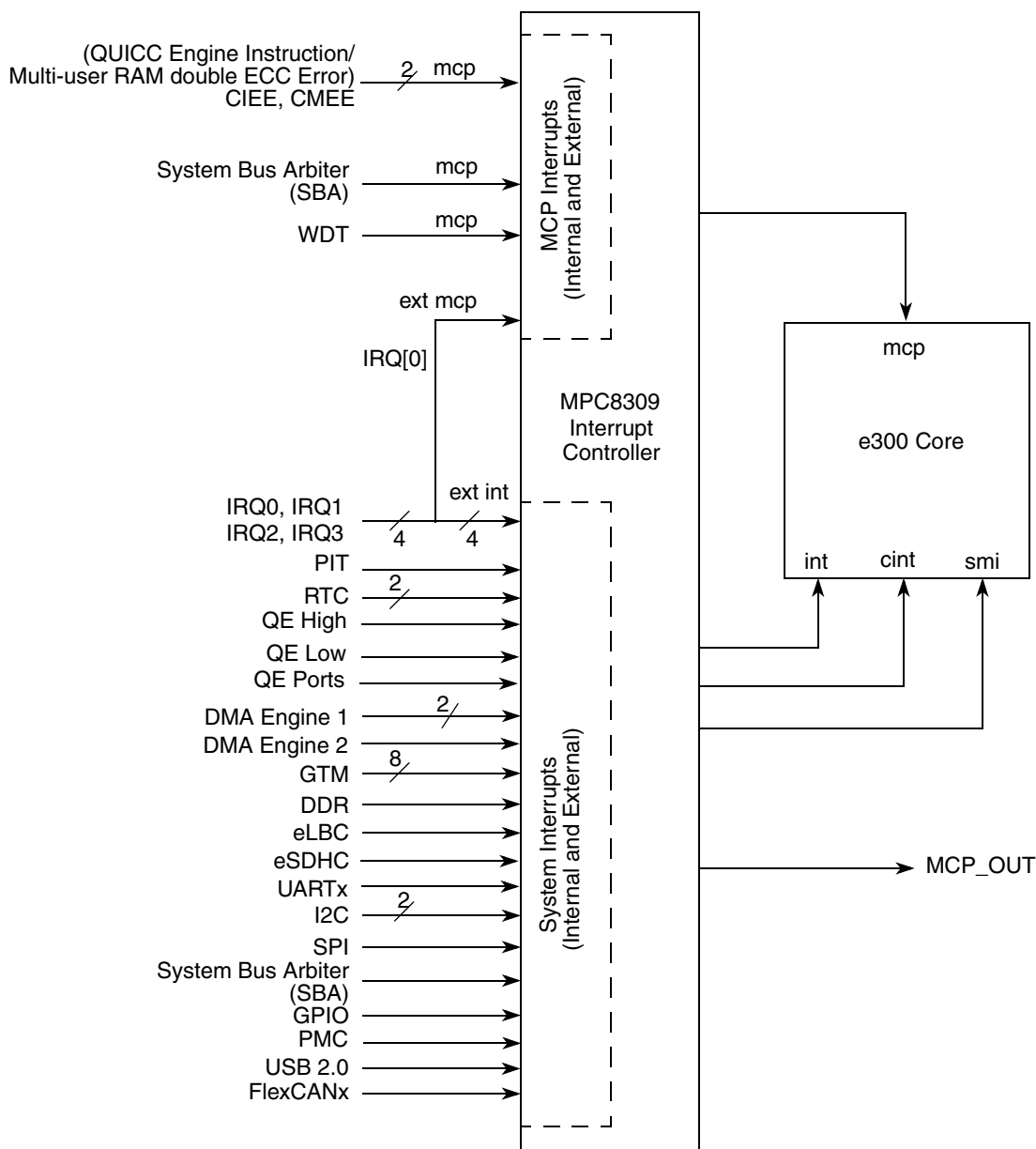


Figure 9-30. Interrupt Structure for MPC8309

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 e300C3 Power Architecture processor 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 , USB DR, QUICC Engine High, and QUICC Engine Low internal interrupt signals can be modified.
- The relative priority of the eSDHC internal interrupt signals can be modified.
- The relative priority of the DMA Engine 1 internal interrupts can be modified.
- The relative priority of the UARTx, FlexCANx, SPI, I2C1, and I2C2, internal interrupt signals can be modified.
- The relative priority of the IRQ0, IRQ1, IRQ2, and IRQ3 external interrupts, and RTC SEC and PIT internal interrupts can be modified.
- The relative priority of the RTC ALR, SBA, and DMA Engine 2 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.

9.6.3 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 (SIPRRx) 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 9-38](#), 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 9-38](#) so other sources can have lower interrupt latencies. This scheme is also programmed but cannot be changed dynamically.

9.6.4 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 (SMPRRx) 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 9-38](#) 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.

9.6.5 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 9-38](#).

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 9-38](#). 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.

9.6.6 Interrupt Source Priorities

Each of the IPIC’s internal and external interrupt sources can independently assert one interrupt request to the core. [Table 9-38](#) 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 9-38. Interrupt Source Priority Levels

Priority	Interrupt Source Description
1	Highest
2	MIXA0 (Grouped/Spread)
3	MIXA1 (Grouped)
4	MIXA2 (Grouped)
5	MIXA3 (Grouped)
6	MIXB0 (Spread)
7	SYSB0 (Grouped)
8	SYSB1 (Grouped)
9	SYSB2 (Grouped)
10	SYSB3 (Grouped)
11	MIXA1 (Spread)
12	SYSB4 (Grouped)
13	SYSB5 (Grouped)
14	SYSB6 (Grouped)
15	SYSB7 (Grouped)
16	MIXB0 (Grouped)
17	MIXB1 (Grouped)
18	MIXB2 (Grouped)
19	MIXB3 (Grouped)
20	MIXB1 (Spread)

Table 9-38. Interrupt Source Priority Levels (continued)

Priority	Interrupt Source Description
21	SYSA0 (Grouped)
22	SYSA1 (Grouped)
23	SYSA2 (Grouped)
24	SYSA3 (Grouped)
25	MIXA2 (Spread)
26	SYSA4 (Grouped)
27	SYSA5 (Grouped)
28	SYSA6 (Grouped)
29	SYSA7 (Grouped)
30	MIXA4 (Grouped)
31	MIXA5 (Grouped)
32	MIXA6 (Grouped)
33	MIXA7 (Grouped)
34	MIXB2 (Spread)
35	SYSC0 (Grouped)
36	SYSC1 (Grouped)
37	SYSC2 (Grouped)
38	SYSC3 (Grouped)
39	MIXA3 (Spread)
40	SYSC4 (Grouped)
41	SYSC5 (Grouped)
42	SYSC6 (Grouped)
43	SYSC7 (Grouped)
44	MIXB4 (Grouped)
45	MIXB5 (Grouped)
46	MIXB6 (Grouped)
47	MIXB7 (Grouped)
48	MIXB3 (Spread)
49	SYSD0 (Grouped)
50	SYSD1 (Grouped)
51	SYSD2 (Grouped)
52	SYSD3 (Grouped)
53	MIXA4 (Spread)
54	SYSD4 (Grouped)
55	SYSD5 (Grouped)
56	SYSD6 (Grouped)
57	SYSD7 (Grouped)

Table 9-38. Interrupt Source Priority Levels (continued)

Priority	Interrupt Source Description
58	MIXB4 (Spread)
59	GTM4
60	SYSB0 (Spread)
61	SYSA0 (Spread)
62	GTM8
63	SYSC0 (Spread)
64	SYSD0 (Spread)
65	Reserved
66	QUICC Engine Ports
67	MIXA5 (Spread)
68	GPIOx
69	SYSB1 (Spread)
70	SYSA1 (Spread)
71	DDR Controller
72	SYSC1 (Spread)
73	SYSD1 (Spread)
74	Reserved
75	eLBC
76	MIXB5 (Spread)
77	GTM2
78	SYSB2 (Spread)
79	SYSA2 (Spread)
80	GTM6
81	SYSC2 (Spread)
82	SYSD2 (Spread)
83	Reserved
84	PMC
85	MIXA6 (Spread)
86	Reserved
87	SYSB3 (Spread)
88	SYSA3 (Spread)
89	Reserved
90	SYSC3 (Spread)
91	SYSD3 (Spread)
92	Reserved
93	Reserved
94	MIXB6 (Spread)

Table 9-38. Interrupt Source Priority Levels (continued)

Priority	Interrupt Source Description
95	GTM3
96	SYSB4 (Spread)
97	SYSA4 (Spread)
98	GTM7
99	SYSC4 (Spread)
100	SYSD4 (Spread)
101	Reserved
102	Reserved
103	MIXA7 (Spread)
104	Reserved
105	SYSB5 (Spread)
106	SYSA5 (Spread)
107	Reserved
108	SYSC5 (Spread)
109	SYSD5 (Spread)
110	Reserved
111	Reserved
112	MIXB7 (Spread)
113	GTM1
114	SYSB6 (Spread)
115	SYSA6 (Spread)
116	GTM5
117	SYSC6 (Spread)
118	SYSD6 (Spread)
119	Reserved
120	Reserved
121	Reserved
122	SYSB7 (Spread)
123	SYSA7 (Spread)
124	DMA Engine 1 ERR
125	SYSC7 (Spread)
126	SYSD7 (Spread)
127	Reserved
128	Reserved

9.6.7 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 9-38 shows which interrupt sources have multiple interrupting events.

Figure 9-31 shows an example of how the masking occurs, using a DDR block as an example.

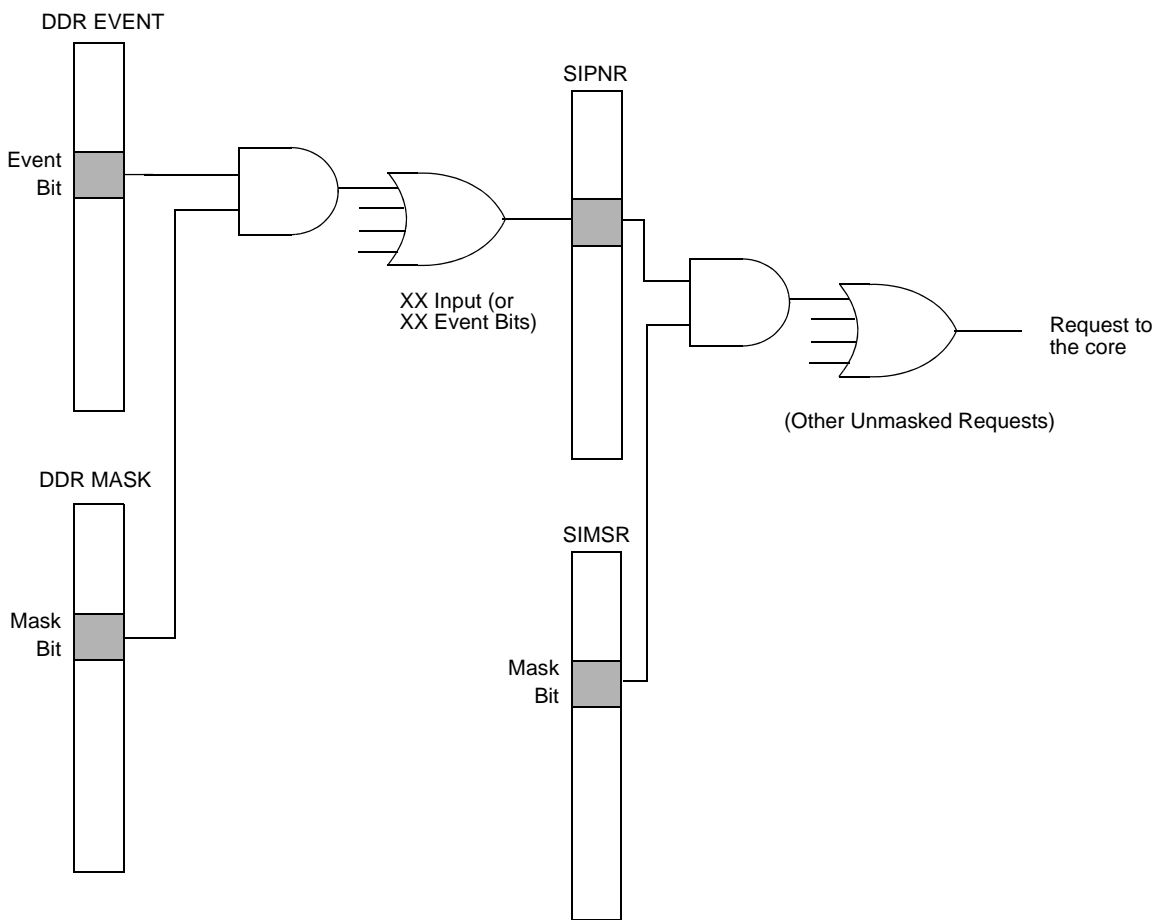


Figure 9-31. DDR Interrupt Request Masking

9.6.8 Interrupt Vector Generation and Calculation

Pending unmasked interrupts are presented to the core in order of priority according to Table 9-38. 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 9-6](#) lists the encodings for the seven low-order bits of the interrupt vector.

9.6.9 Machine Check Interrupts

The IPIC supports the non-maskable machine check interrupts. When an error interrupt signal is received, the interrupt controller indicates the source by setting the corresponding SERSR bit. These sources are listed in [Table 9-24](#).



Chapter 10

DDR Memory Controller

10.1 Introduction

The fully programmable DDR SDRAM controller supports most JEDEC standard $\times 8$, and $\times 16$, and $\times 32$ DDR2 memories available. In addition, unbuffered and registered DRAM modules are supported. However, mixing different memory types or unbuffered and registered DRAM modules in the same system is not supported. Dynamic power management and auto-precharge modes simplify memory system design. A large set of special features support rapid system debug.

NOTE

- In this chapter, the word ‘bank’ refers to a physical bank specified by a chip select; ‘logical bank’ refers to one of the four or eight sub-banks in each SDRAM chip. A sub-bank is specified by the 2 or 3 bits on the bank address (MBA) pins during a memory access.
- MPC8309 supports only DDR2 controller. In this chapter, usage of the word ‘DDR’ is generic and refers to the DDR2 controller.

Figure 10-1 is a high-level block diagram of the DDR memory controller with its associated interfaces.

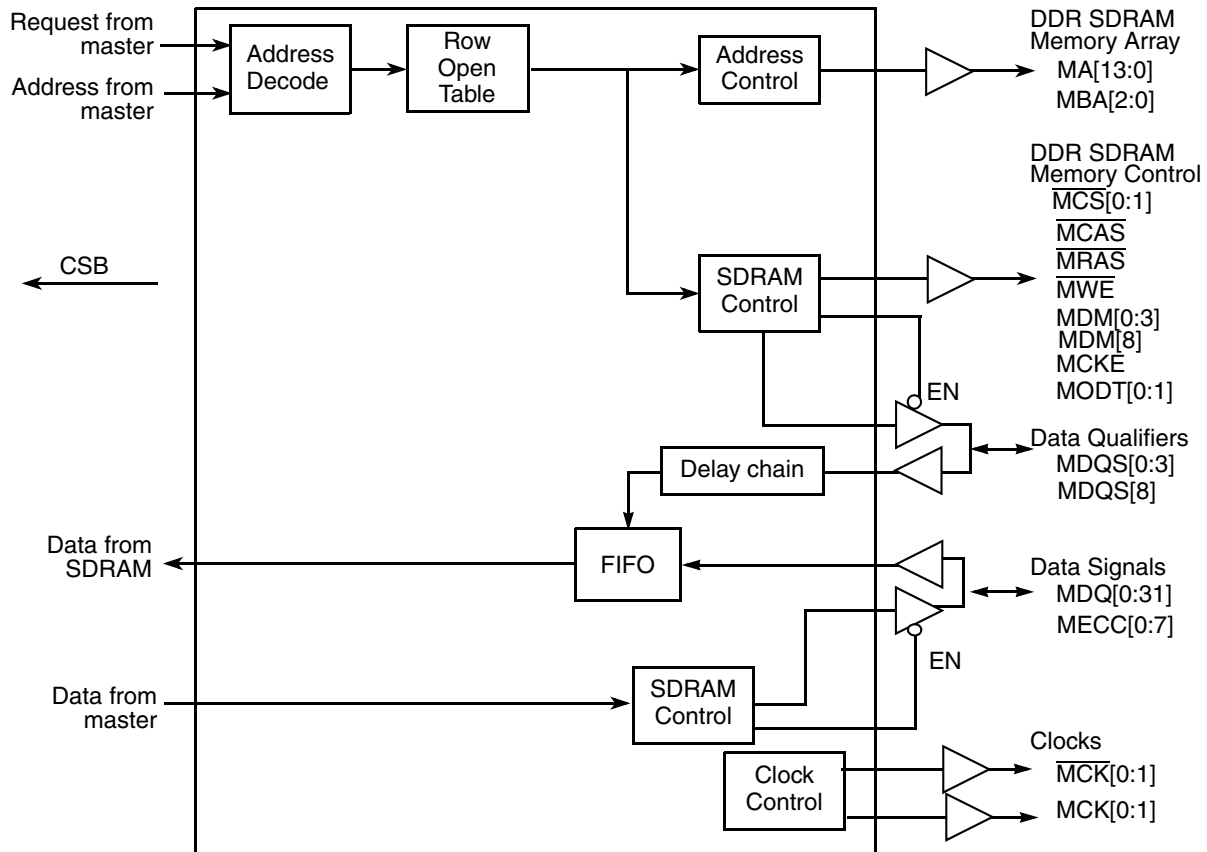


Figure 10-1. DDR Memory Controller Simplified Block Diagram

10.2 Features

The DDR memory controller includes these distinctive features:

- Support for DDR2 SDRAM
- Supports 8-bit ECC
- Supports 16-/32-bit data interface
- Programmable settings for meeting all SDRAM timing parameters
- The following SDRAM configurations are supported:
 - As many as two physical banks (chip selects), each bank independently addressable up to 512 Mbytes
 - 64-Mbit to 2-Gbit devices depending on internal device configuration with $\times 8/\times 16/\times 32$ data ports (no direct $\times 4$ support)
 - One 32-bit device, two 16-bit device or four 8-bit devices on a 32-bit bus
 - Unbuffered and registered
- 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

10.2.1 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 `CSn_CONFIG[AP_n_EN]`.

10.3 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.

10.3.1 Signals Overview

Memory controller signals are grouped as follows:

- Memory interface signals
- Clock signals
- Debug signals

Table 10-1 shows how DDR memory controller external signals are grouped. The *MPC8309 PowerQUICC II Pro Integrated Communications Processor Hardware Specification* has a pinout diagram showing pin numbers. It also lists all electrical and mechanical specifications.

Table 10-1. DDR Memory Interface Signal Summary

Name	Function/Description	Reset	Pins	I/O
MDQ[0:31]	DDR data bus	All zeros	32	I/O
MDQS[0:3]	DDR data strobe	All zeros	4	I/O

Table 10-1. DDR Memory Interface Signal Summary (continued)

Name	Function/Description	Reset	Pins	I/O
MDM[0:3]	DDR data mask	All zeros	4	O
MECC[0:7]	DDR ECC bits	All zeros	8	I/O
MDQS[8]	DDR data strobe	All zeros	1	I/O
MDM[8]	DDR data mask	All zeros	1	O
MA[0:13]	DDR address bus	All zeros	14	O
MBA[0:2]	DDR bank select	All zeros	3	O
$\overline{\text{MCAS}}$	DDR column address strobe	One	1	O
$\overline{\text{MRAS}}$	DDR row address strobe	One	1	O
$\overline{\text{MWE}}$	DDR write enable	One	1	O
$\overline{\text{MCS}}[0:1]$	DDR chip select	All ones	2	O
MCK[0:1]	DDR differential clocks	Zero	2	O
$\overline{\text{MCK}}[0:1]$	DDR differential clocks	One	2	O
MCKE	DDR clock enable	Zero	2	O
MODT[0:1]	DDR on-die termination	All zeros	2	O
MVREF	DDR DRAM reference	—	1	PWR

Table 10-2 shows the memory address signal mappings.

Table 10-2. Memory Address Signal Mappings

Signal Name (Outputs)		JEDEC DDR DIMM Signals (Inputs)
msb	MA13	A13
	MA12	A12
	MA11	A11
	MA10	A10 (AP for DDR) ¹
	MA9	A9
	MA8	A8 (alternate AP for DDR) ²
	MA7	A7
	MA6	A6
	MA5	A5
	MA4	A4
	MA3	A3
	MA2	A2
	MA1	A1
	lsb	MA0
msb	MBA2	MBA2
	MBA1	MBA1
lsb	MBA0	MBA0

¹ Auto-precharge for DDR signaled on A10 when DDR_SDRAM_CFG[PCHB8] = 0

² Auto-precharge for DDR signaled on A8 when DDR_SDRAM_CFG[PCHB8] = 1

10.3.2 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.

10.3.2.1 Memory Interface Signals

Table 10-3 describes the DDR controller memory interface signals.

Table 10-3. Memory Interface Signals—Detailed Signal Descriptions

Signal	I/O	Description	
MDQ[0:31]	I/O	Data bus. Both input and output signals on the DDR memory controller.	
	O	As outputs for the bidirectional data bus, these signals operate as described below.	
		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 as described below.	
		State Meaning	Asserted/Negated—Represents the state of data being driven by the external DDR SDRAMs.
Timing		Assertion/Negation—The DDR SDRAM drives data during a READ transaction. High impedance—No READ or WRITE command in progress; data is not being driven by the memory controller or the DRAM.	
MDQS[0:3]	I/O	Data strobes. Inputs with read data, outputs with write data.	
		O	As outputs, the data strobes are driven by the DDR memory controller during a write transaction. The memory controller always drives these signals low unless a read has been issued and incoming data strobes are expected. This keeps the data strobes from floating high when there are no transactions on the DRAM interface.
			State Meaning
	Timing	Assertion/Negation—If a WRITE command is registered at clock edge, data strobes at the DRAM assert centered in the data eye. For more information, see the JEDEC DDR2 SDRAM specification.	
	I	As inputs, the data strobes are driven by the external DDR SDRAMs during a read transaction. The data strobes are used by the memory controller to synchronize data latching.	
		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 10-36 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 10-3. Memory Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
MA[13:0]	O	Address bus. Memory controller outputs for the address to the DRAM. MA[13:0] carry 14 of the address bits for the DDR memory interface corresponding to the row and column address bits. MA0 is the lsb of the address output from the memory controller.	
		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 10-38 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{MCSn} 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 10-38 describes the mapping of these signals in all cases.
		Timing	Assertion/Negation—Same timing as MA_n High impedance—Same timing as MA_n
\overline{MCAS}	O	Column address strobe. Active-low SDRAM address multiplexing signal. \overline{MCAS} is asserted for read or write transactions and for mode register set, refresh, and precharge commands.	
		State Meaning	Asserted—Indicates that a valid SDRAM column address is on the address bus for read and write transactions. See Table 10-41 for more information on the states required on \overline{MCAS} for various other SDRAM commands. Negated—The column address is not guaranteed to be valid.
		Timing	Assertion/Negation—Assertion and negation timing is directed by the values described in Section 10.4.1.4, "DDR SDRAM Timing Configuration 0 (TIMING_CFG_0)," Section 10.4.1.5, "DDR SDRAM Timing Configuration 1 (TIMING_CFG_1)," Section 10.4.1.6, "DDR SDRAM Timing Configuration 2 (TIMING_CFG_2)," and Section 10.4.1.3, "DDR SDRAM Timing Configuration 3 (TIMING_CFG_3)." High impedance— \overline{MCAS} is always driven unless the memory controller is disabled.
\overline{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 10-41 for more information on the states required on \overline{MRAS} for various other SDRAM commands. Negated—The row address is not guaranteed to be valid.
		Timing	Assertion/Negation—Assertion and negation timing is directed by the values described in Section 10.4.1.4, "DDR SDRAM Timing Configuration 0 (TIMING_CFG_0)," Section 10.4.1.5, "DDR SDRAM Timing Configuration 1 (TIMING_CFG_1)," Section 10.4.1.6, "DDR SDRAM Timing Configuration 2 (TIMING_CFG_2)," and Section 10.4.1.3, "DDR SDRAM Timing Configuration 3 (TIMING_CFG_3)." High impedance— \overline{MRAS} is always driven unless the memory controller is disabled.

Table 10-3. Memory Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description
$\overline{\text{MCS}}[0:1]$	O	Chip selects. Two chip selects are supported by the memory controller.
		State Meaning Asserted—Selects a physical SDRAM bank to perform a memory operation as described in Section 10.4.1.1 , “Chip Select Memory Bounds ($\text{CS}_n\text{_BND}$ S),” and Section 10.4.1.2 , “Chip Select Configuration ($\text{CS}_n\text{_CONFIG}$).” The DDR controller asserts one of the $\overline{\text{MCS}}[0:1]$ 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 10-41 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 $\overline{\text{MRA}}\text{S}$ and $\overline{\text{MCAS}}$. Used for write commands. High impedance— $\overline{\text{MWE}}$ is always driven unless the memory controller is disabled.
MDM[8]	O	DDR SDRAM data output mask. Masks unwanted bytes of data transferred during a write. They are needed to support sub-burst-size transactions (such as single-byte writes) on SDRAM where all I/O occurs in multi-byte bursts. MDM_0 corresponds to the most significant byte (MSB). Table 10-36 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 written to the SDRAM.
		Timing Assertion/Negation—Same timing as MDQx as outputs. High-impedance—Always driven unless the memory controller is disabled.
MODT[0:1]	O	On-Die termination. Memory controller outputs for the ODT to the DRAM. MODT[0:1] represents the on-die termination for the associated data, data masks, 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 $\text{CS}_n\text{_CONFIG}[\text{ODT_RD_CFG}]$ and $\text{CS}_n\text{_CONFIG}[\text{ODT_WR_CFG}]$ fields. High impedance—Always driven.

10.3.2.2 Clock Interface Signals

[Table 10-4](#) contains the detailed descriptions of the clock signals of the DDR controller.

Table 10-4. Clock Signals—Detailed Signal Descriptions

Signal	I/O	Description
$\text{MCK}[0:1]$, $\overline{\text{MCK}}[0:1]$	O	DRAM clock output and its complement. See Section 10.5.4.1 , “Clock Distribution.”
		State Meaning Asserted/Negated—The JEDEC DDR SDRAM specifications require true and complement clocks. A clock edge is seen by the SDRAM when the true and complement cross.
		Timing Assertion/Negation—Timing is controlled by the DDR_CLK_CNTL register at offset 0x130.

Table 10-4. Clock Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description
MCKE	O	Clock enable. Output signals used as the clock enables to the SDRAM. MCKE can be negated to stop clocking the DDR SDRAM.
		State Asserted—Clocking to the SDRAM is enabled. Meaning Negated—Clocking to the SDRAM is disabled and the SDRAM should ignore signal transitions on MCK0 or $\overline{\text{MCK0}}$. MCK0/ $\overline{\text{MCK0}}$ are don't cares while MCKE is negated.
		Timing Assertion/Negation—Asserted when DDR_SDRAM_CFG[MEM_EN] is set. Can be negated when entering dynamic power management or self refresh. Are asserted again when exiting dynamic power management or self refresh. High impedance—Always driven.

10.4 Memory Map/Register Definition

Table 10-5 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 10-5. DDR Memory Controller Memory Map

Offset	Register	Access	Reset	Section/Page
DDR Memory Controller—Block Base Address 0x0_2000				
0x000	CS0_BNDS—Chip select 0 memory bounds	R/W	0x0000_0000	10.4.1.1/10-11
0x008	CS1_BNDS—Chip select 1 memory bounds	R/W	0x0000_0000	10.4.1.1/10-11
0x080	CS0_CONFIG—Chip select 0 configuration	R/W	0x0000_0000	10.4.1.2/10-11
0x084	CS1_CONFIG—Chip select 1 configuration	R/W	0x0000_0000	10.4.1.2/10-11
0x100	TIMING_CFG_3—DDR SDRAM timing configuration 3	R/W	0x0000_0000	10.4.1.3/10-13
0x104	TIMING_CFG_0—DDR SDRAM timing configuration 0	R/W	0x0011_0105	10.4.1.4/10-14
0x108	TIMING_CFG_1—DDR SDRAM timing configuration 1	R/W	0x0000_0000	10.4.1.5/10-16
0x10C	TIMING_CFG_2—DDR SDRAM timing configuration 2	R/W	0x0000_0000	10.4.1.6/10-18
0x110	DDR_SDRAM_CFG—DDR SDRAM control configuration	R/W	0x0200_0000	10.4.1.7/10-19
0x114	DDR_SDRAM_CFG_2—DDR SDRAM control configuration 2	R/W	0x0000_0000	10.4.1.8/10-22
0x118	DDR_SDRAM_MODE—DDR SDRAM mode configuration	R/W	0x0000_0000	10.4.1.9/10-23
0x11C	DDR_SDRAM_MODE_2—DDR SDRAM mode configuration 2	R/W	0x0000_0000	10.4.1.10/10-24
0x120	DDR_SDRAM_MD_CNTL—DDR SDRAM mode control	R/W	0x0000_0000	10.4.1.11/10-25

Table 10-5. DDR Memory Controller Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x124	DDR_SDRAM_INTERVAL—DDR SDRAM interval configuration	R/W	0x0000_0000	10.4.1.12/10-27
0x128	DDR_DATA_INIT—DDR SDRAM data initialization	R/W	0x0000_0000	10.4.1.13/10-28
0x130	DDR_SDRAM_CLK_CNTL—DDR SDRAM clock control	R/W	0x0200_0000	10.4.1.14/10-28
0x140–0x144	Reserved	—	—	—
0x148	DDR_INIT_ADDR—DDR training initialization address	R/W	0x0000_0000	10.4.1.15/10-29
0x150–0xBF4	Reserved	—	—	—
0xBF8	DDR_IP_REV1—DDR IP block revision 1	R	0xnnnn_nnnn ¹	10.4.1.16/10-29
0xBFC	DDR_IP_REV2—DDR IP block revision 2	R	0x00nn_00nn	10.4.1.17/10-30
0xE00	DATA_ERR_INJECT_HI—Memory data path error injection mask high	R/W	0x0000_0000	10.4.1.18/10-30
0xE04	DATA_ERR_INJECT_LO—Memory data path error injection mask low	R/W	0x0000_0000	10.4.1.19/10-31
0xE08	ERR_INJECT—Memory data path error injection mask ECC	R/W	0x0000_0000	10.4.1.20/10-31
0xE20	CAPTURE_DATA_HI—Memory data path read capture high	R/W	0x0000_0000	10.4.1.21/10-32
0xE24	CAPTURE_DATA_LO—Memory data path read capture low	R/W	0x0000_0000	10.4.1.22/10-32
0xE28	CAPTURE_ECC—Memory data path read capture ECC	R/W	0x0000_0000	10.4.1.23/10-33
0xE40	ERR_DETECT—Memory error detect	w1c	0x0000_0000	10.4.1.24/10-33
0xE44	ERR_DISABLE—Memory error disable	R/W	0x0000_0000	10.4.1.25/10-34
0xE48	ERR_INT_EN—Memory error interrupt enable	R/W	0x0000_0000	10.4.1.26/10-35
0xE4C	CAPTURE_ATTRIBUTES—Memory error attributes capture	R/W	0x0000_0000	10.4.1.27/10-36
0xE50	CAPTURE_ADDRESS—Memory error address capture	R/W	0x0000_0000	10.4.1.28/10-37
0xE54	Reserved	—	—	—
0xE58	ERR_SBE—Single-Bit ECC memory error management	R/W	0x0000_0000	10.4.1.29/10-37

¹ Implementation-dependent reset values are listed in specified section/page.

10.4.1 Register Descriptions

This section describes the DDR memory controller registers. Shading indicates reserved fields that should not be written.

10.4.1.1 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 EAn must be greater than or equal to SAn .

If chip select interleaving is enabled, all fields in the lower interleaved chip select are used, and the other chip selects' bounds registers are unused. For example, if chip selects 0 and 1 are interleaved, all fields in CS_0_BNDS are used, and all fields in CS_1_BNDS are unused.

CS_n_BNDS are shown in [Figure 10-2](#).



Figure 10-2. Chip Select Bounds Registers (CS_n_BNDS)

[Table 10-6](#) describes the CS_n_BNDS register fields.

Table 10-6. CS_n_BNDS Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8–15	SAn	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	EAn	Ending address for chip select (bank) n . This value is compared against the 8 msbs of the 32-bit address.

10.4.1.2 Chip Select Configuration (CS_n_CONFIG)

The chip select configuration (CS_n_CONFIG) registers shown in [Figure 10-3](#) enable the DDR chip selects and set the number of row and column bits used for each chip select. These registers should be loaded with the correct number of row and column bits for each SDRAM. Because $CS_n_CONFIG[ROW_BITS_CS_n, COL_BITS_CS_n]$ establish address multiplexing, the user should take great care to set these values correctly.

If chip select interleaving is enabled, then all fields in the lower interleaved chip select are used, and the other registers' fields are unused, with the exception of the ODT_RD_CFG and ODT_WR_CFG fields.

For example, if chip selects 0 and 1 are interleaved, all fields in CS0_CONFIG are used, but only the ODT_RD_CFG and ODT_WR_CFG fields in CS1_CONFIG are used.

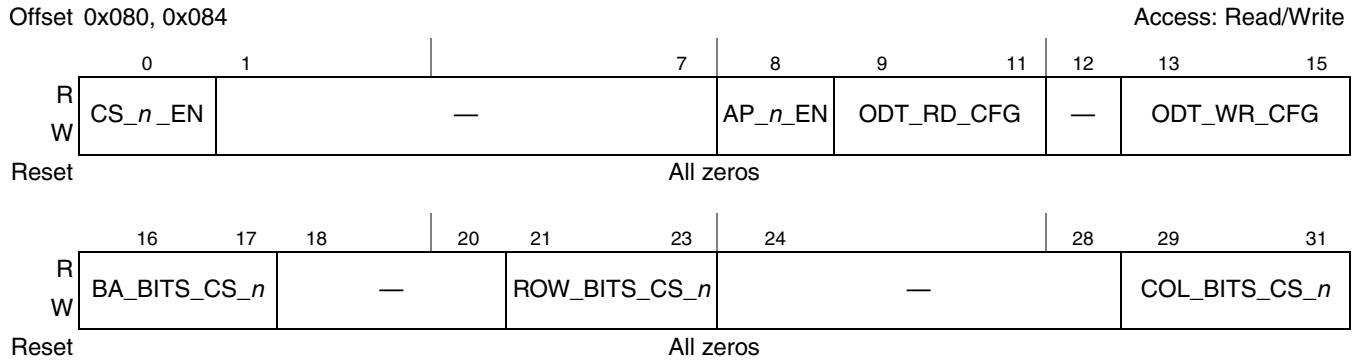


Figure 10-3. Chip Select Configuration Register (CS_n_CONFIG)

Table 10-7 describes the CS_n_CONFIG register fields.

Table 10-7. CS_n_CONFIG Field Descriptions

Bits	Name	Description
0	CS _n _EN	Chip select <i>n</i> enable 0 Chip select <i>n</i> is not active 1 Chip select <i>n</i> is active and assumes the state set in CS _n _BNDS.
1–7	—	Reserved
8	AP _n _EN	Chip select <i>n</i> auto-precharge enable 0 Chip select <i>n</i> is only auto-precharged if global auto-precharge mode is enabled (DDR_SDRAM_INTERVAL[BSTOPRE] = 0). 1 Chip select <i>n</i> always issues an auto-precharge for read and write transactions.
9–11	ODT_RD_CFG	ODT for reads configuration. Note that CAS latency plus additive latency must be at least 3 cycles for ODT_RD_CFG to be enabled. 000 Never assert ODT for reads 001 Assert ODT only during reads to CS _n 010 Assert ODT only during reads to other chip selects 011 Reserved 100 Assert ODT for all reads 101–111Reserved
12	—	Reserved
13–15	ODT_WR_CFG	ODT for writes configuration. Note that write latency plus additive latency must be at least 3 cycles for ODT_WR_CFG to be enabled. 000 Never assert ODT for writes 001 Assert ODT only during writes to CS _n 010 Assert ODT only during writes to other chip selects 011 Reserved 100 Assert ODT for all writes 101–111Reserved
16–17	BA_BITS_CS _n	Number of bank bits for SDRAM on chip select <i>n</i> . These bits correspond to the sub-bank bits driven on MBAn in Table 10-38 . 00 2 logical bank bits 01 3 logical bank bits 10–11Reserved

Table 10-7. CS_n_CONFIG Field Descriptions (continued)

Bits	Name	Description
18–20	—	Reserved
21–23	ROW_BITS_CS _n	Number of row bits for SDRAM on chip select <i>n</i> . See Table 10-38 for details. 000 12 row bits 001 13 row bits 010 14 row bits 011–111 Reserved
24–28	—	Reserved
29–31	COL_BITS_CS _n	Number of column bits for SDRAM on chip select <i>n</i> . For DDR, the decoding is as follows: 000 8 column bits 001 9 column bits 010 10 column bits 011 11 column bits 100–111 Reserved

10.4.1.3 DDR SDRAM Timing Configuration 3 (TIMING_CFG_3)

DDR SDRAM timing configuration register 3, shown in [Figure 10-4](#), sets the extended refresh recovery time, which is combined with TIMING_CFG_1[REFREC] to determine the full refresh recovery time.

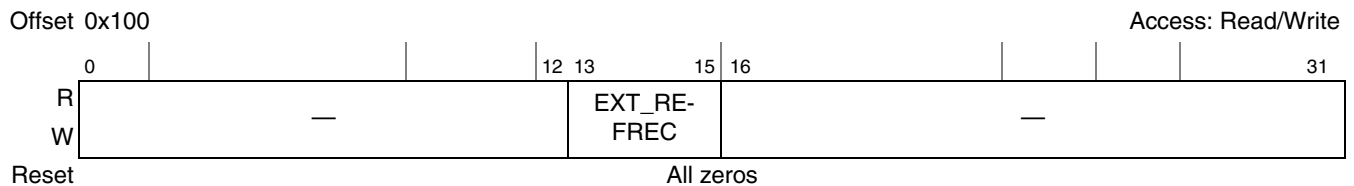


Figure 10-4. DDR SDRAM Timing Configuration 3 (TIMING_CFG_3)

[Table 10-8](#) describes TIMING_CFG_3 fields.

Table 10-8. TIMING_CFG_3 Field Descriptions

Bits	Name	Description
0–12	—	Reserved, should be cleared.
13–15	EXT_REFREC	Extended refresh recovery time (t_{RFC}). Controls the number of clock cycles from a refresh command until an activate command is allowed. This field is concatenated with TIMING_CFG_1[REFREC] to obtain a 7-bit value for the total refresh recovery. Note that hardware adds an additional 8 clock cycles to the final, 7-bit value of the refresh recovery. $t_{RFC} = \{EXT_REFREC \parallel REFREC\} + 8$, such that t_{RFC} is calculated as follows: 000 0 clocks 001 16 clocks 010 32 clocks 011 48 clocks 100 64 clocks 101 80 clocks 110 96 clocks 111 112 clocks
16–31	—	Reserved, should be cleared.



10.4.1.4 DDR SDRAM Timing Configuration 0 (TIMING_CFG_0)

DDR SDRAM timing configuration register 0, shown in [Figure 10-5](#), 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	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	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 10-5. DDR SDRAM Timing Configuration 0 (TIMING_CFG_0)

[Table 10-9](#) describes TIMING_CFG_0 fields.

Table 10-9. TIMING_CFG_0 Field Descriptions

Bits	Name	Description								
0–1	RWT	Read-to-write turnaround (t_{RTW}). Specifies how many extra cycles are added between a read to write turnaround. If 0 clocks is chosen, then the DDR controller uses a fixed number based on the CAS latency and write latency. Choosing a value other than 0 adds extra cycles past this default calculation. As a default the DDR controller determines the read-to-write turnaround as $CL - WL + BL \div 2 + 2$. In this equation, CL is the CAS latency rounded up to the next integer, WL is the programmed write latency, and BL is the burst length. <table border="0"> <tr> <td>00</td> <td>0 clocks</td> <td>10</td> <td>2 clocks</td> </tr> <tr> <td>01</td> <td>1 clock</td> <td>11</td> <td>3 clocks</td> </tr> </table>	00	0 clocks	10	2 clocks	01	1 clock	11	3 clocks
00	0 clocks	10	2 clocks							
01	1 clock	11	3 clocks							
2–3	WRT	Write-to-read turnaround. Specifies how many extra cycles are added between a write to read turnaround. If 0 clocks is chosen, then the DDR controller uses a fixed number based on the, read latency, and write latency. Choosing a value other than 0 adds extra cycles past this default calculation. As a default, the DDR controller determines the write-to-read turnaround as $WL - CL + BL \div 2 + 1$. In this equation, CL is the CAS latency rounded down to the next integer, WL is the programmed write latency, and BL is the burst length. <table border="0"> <tr> <td>00</td> <td>0 clocks</td> <td>10</td> <td>2 clocks</td> </tr> <tr> <td>01</td> <td>1 clock</td> <td>11</td> <td>3 clocks</td> </tr> </table>	00	0 clocks	10	2 clocks	01	1 clock	11	3 clocks
00	0 clocks	10	2 clocks							
01	1 clock	11	3 clocks							
4–5	RRT	Read-to-read turnaround. Specifies how many extra cycles are added between reads to different chip selects. As a default, 3 cycles are required between read commands to different chip selects. Extra cycles may be added with this field. Note: If 8-beat bursts are enabled, then 5 cycles are the default. Note that DDR2 does not support 8-beat bursts. <table border="0"> <tr> <td>00</td> <td>0 clocks</td> <td>10</td> <td>2 clocks</td> </tr> <tr> <td>01</td> <td>1 clock</td> <td>11</td> <td>3 clocks</td> </tr> </table>	00	0 clocks	10	2 clocks	01	1 clock	11	3 clocks
00	0 clocks	10	2 clocks							
01	1 clock	11	3 clocks							
6–7	WWT	Write-to-write turnaround. Specifies how many extra cycles are added between writes to different chip selects. As a default, 2 cycles are required between write commands to different chip selects. Extra cycles may be added with this field. Note: If 8-beat bursts are enabled, then 4 cycles are the default. Note that DDR2 does not support 8-beat bursts. <table border="0"> <tr> <td>00</td> <td>0 clocks</td> <td>10</td> <td>2 clocks</td> </tr> <tr> <td>01</td> <td>1 clock</td> <td>11</td> <td>3 clocks</td> </tr> </table>	00	0 clocks	10	2 clocks	01	1 clock	11	3 clocks
00	0 clocks	10	2 clocks							
01	1 clock	11	3 clocks							
8	—	Reserved, should be cleared.								

Table 10-9. 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 100 4 clocks 001 1 clock 101 5 clocks 010 2 clocks 110 6 clocks 011 3 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 1000 8 clocks 0001 1 clock 1001 9 clocks 0010 2 clocks 1010 10 clocks 0011 3 clocks 1011 11 clocks 0100 4 clocks 1100 12 clocks 0101 5 clocks 1101 13 clocks 0110 6 clocks 1110 14 clocks 0111 7 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 1000 8 clocks 0001 1 clock 1001 9 clocks 0010 2 clocks 1010 10 clocks 0011 3 clocks 1011 11 clocks 0100 4 clocks 1100 12 clocks 0101 5 clocks 1101 13 clocks 0110 6 clocks 1110 14 clocks 0111 7 clocks 1111 15 clocks

10.4.1.5 DDR SDRAM Timing Configuration 1 (TIMING_CFG_1)

DDR SDRAM timing configuration register 1, shown in Figure 10-6, sets the number of clock cycles between various SDRAM control commands.

Offset 0x108

Access: Read/Write



Figure 10-6. DDR SDRAM Timing Configuration 1 (TIMING_CFG_1)

Table 10-10 describes TIMING_CFG_1 fields.

Table 10-10. TIMING_CFG_1 Field Descriptions

Bits	Name	Description
0	—	Reserved, should be cleared.
1–3	PRETOACT	Precharge-to-activate interval (t_{RP}). Determines the number of clock cycles from a precharge command until an activate or refresh command is allowed. 000 Reserved 001 1 clock 010 2 clocks 011 3 clocks 100 4 clocks 101 5 clocks 110 6 clocks 111 7 clocks
4–7	ACTTOPRE	Activate to precharge interval (t_{RAS}). Determines the number of clock cycles from an activate command until a precharge command is allowed. 0000 16 clocks 0101 5 clocks 0001 17 clocks 0110 6 clocks 0010 18 clocks 0111 7 clocks 0011 19 clocks ... 0100 4 clocks 1111 15 clocks
8	—	Reserved, should be cleared.
9–11	ACTTORW	Activate to read/write interval for SDRAM (t_{RCD}). Controls the number of clock cycles from an activate command until a read or write command is allowed. 000 Reserved 001 1 clock 010 2 clocks 011 3 clocks 100 4 clocks 101 5 clocks 110 6 clocks 111 7 clocks

Table 10-10. TIMING_CFG_1 Field Descriptions (continued)

Bits	Name	Description																																
12–15	CASLAT	<p>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 10.4.1.8, “DDR SDRAM Control Configuration 2 (DDR_SDRAM_CFG_2).”</p> <table> <tr><td>0000</td><td>Reserved</td><td>1000</td><td>4.5 clocks</td></tr> <tr><td>0001</td><td>Reserved</td><td>1001</td><td>5 clocks</td></tr> <tr><td>0010</td><td>Reserved</td><td>1010</td><td>5.5 clocks</td></tr> <tr><td>0011</td><td>Reserved</td><td>1011</td><td>6 clocks</td></tr> <tr><td>0100</td><td>Reserved</td><td>1100</td><td>6.5 clocks</td></tr> <tr><td>0101</td><td>3 clocks</td><td>1101</td><td>7 clocks</td></tr> <tr><td>0110</td><td>3.5 clocks</td><td>1110</td><td>7.5 clocks</td></tr> <tr><td>0111</td><td>4 clocks</td><td>1111</td><td>8 clocks</td></tr> </table>	0000	Reserved	1000	4.5 clocks	0001	Reserved	1001	5 clocks	0010	Reserved	1010	5.5 clocks	0011	Reserved	1011	6 clocks	0100	Reserved	1100	6.5 clocks	0101	3 clocks	1101	7 clocks	0110	3.5 clocks	1110	7.5 clocks	0111	4 clocks	1111	8 clocks
0000	Reserved	1000	4.5 clocks																															
0001	Reserved	1001	5 clocks																															
0010	Reserved	1010	5.5 clocks																															
0011	Reserved	1011	6 clocks																															
0100	Reserved	1100	6.5 clocks																															
0101	3 clocks	1101	7 clocks																															
0110	3.5 clocks	1110	7.5 clocks																															
0111	4 clocks	1111	8 clocks																															
16–19	REFREC	<p>Refresh recovery time (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: $t_{RFC} = \{EXT_REFREC \parallel REFREC\} + 8$.</p> <table> <tr><td>0000</td><td>8 clocks</td><td>0011</td><td>11 clocks</td></tr> <tr><td>0001</td><td>9 clocks</td><td>...</td><td></td></tr> <tr><td>0010</td><td>10 clocks</td><td>1111</td><td>23 clocks</td></tr> </table>	0000	8 clocks	0011	11 clocks	0001	9 clocks	...		0010	10 clocks	1111	23 clocks																				
0000	8 clocks	0011	11 clocks																															
0001	9 clocks	...																																
0010	10 clocks	1111	23 clocks																															
20	—	Reserved, should be cleared.																																
21–23	WRREC	<p>Last data to precharge minimum interval (t_{WR}). Determines the number of clock cycles from the last data associated with a write command until a precharge command is allowed.</p> <table> <tr><td>000</td><td>Reserved</td></tr> <tr><td>001</td><td>1 clock</td></tr> <tr><td>010</td><td>2 clocks</td></tr> <tr><td>011</td><td>3 clocks</td></tr> <tr><td>100</td><td>4 clocks</td></tr> <tr><td>101</td><td>5 clocks</td></tr> <tr><td>110</td><td>6 clocks</td></tr> <tr><td>111</td><td>7 clocks</td></tr> </table>	000	Reserved	001	1 clock	010	2 clocks	011	3 clocks	100	4 clocks	101	5 clocks	110	6 clocks	111	7 clocks																
000	Reserved																																	
001	1 clock																																	
010	2 clocks																																	
011	3 clocks																																	
100	4 clocks																																	
101	5 clocks																																	
110	6 clocks																																	
111	7 clocks																																	
24	—	Reserved, should be cleared.																																
25–27	ACTTOACT	<p>Activate-to-activate interval (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).</p> <table> <tr><td>000</td><td>Reserved</td><td>100</td><td>4 clocks</td></tr> <tr><td>001</td><td>1 clock</td><td>101</td><td>5 clocks</td></tr> <tr><td>010</td><td>2 clocks</td><td>110</td><td>6 clocks</td></tr> <tr><td>011</td><td>3 clocks</td><td>111</td><td>7 clocks</td></tr> </table>	000	Reserved	100	4 clocks	001	1 clock	101	5 clocks	010	2 clocks	110	6 clocks	011	3 clocks	111	7 clocks																
000	Reserved	100	4 clocks																															
001	1 clock	101	5 clocks																															
010	2 clocks	110	6 clocks																															
011	3 clocks	111	7 clocks																															
28	—	Reserved, should be cleared.																																
29–31	WRTORD	<p>Last write data pair to read command issue interval (t_{WTR}). Number of clock cycles between the last write data pair and the subsequent read command to the same physical bank.</p> <table> <tr><td>000</td><td>Reserved</td><td>100</td><td>4 clocks</td></tr> <tr><td>001</td><td>1 clock</td><td>101</td><td>5 clocks</td></tr> <tr><td>010</td><td>2 clocks</td><td>110</td><td>6 clocks</td></tr> <tr><td>011</td><td>3 clocks</td><td>111</td><td>7 clocks</td></tr> </table>	000	Reserved	100	4 clocks	001	1 clock	101	5 clocks	010	2 clocks	110	6 clocks	011	3 clocks	111	7 clocks																
000	Reserved	100	4 clocks																															
001	1 clock	101	5 clocks																															
010	2 clocks	110	6 clocks																															
011	3 clocks	111	7 clocks																															

10.4.1.6 DDR SDRAM Timing Configuration 2 (TIMING_CFG_2)

DDR SDRAM timing configuration 2, shown in [Figure 10-7](#), sets the clock delay to data for writes.

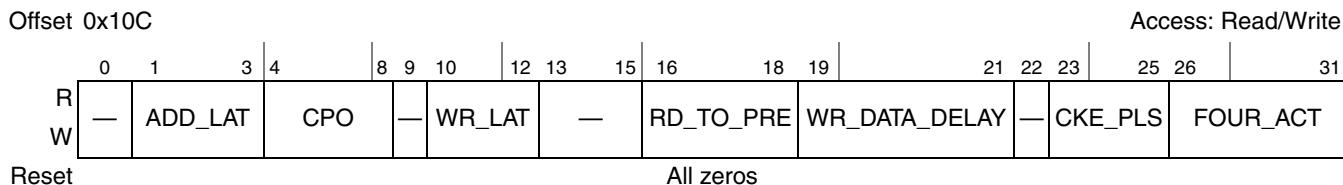


Figure 10-7. DDR SDRAM Timing Configuration 2 Register (TIMING_CFG_2)

[Table 10-11](#) describes the TIMING_CFG_2 fields.

Table 10-11. TIMING_CFG_2 Field Descriptions

Bits	Name	Description
0	—	Reserved
1–3	ADD_LAT	Additive latency. The additive latency must be set to a value less than TIMING_CFG_1[ACTTORW]. 000 0 clocks 001 1 clock 010 2 clocks 011 3 clocks 100 4 clocks 101 5 clocks 110 Reserved 111 Reserved
4–8	CPO	MCAS-to-preamble override. Defines the number of DRAM cycles between when a read is issued and when the corresponding DQS preamble is valid for the memory controller. For these decodings, “READ_LAT” is equal to the CAS latency plus the additive latency. 00000READ_LAT + 1 01100READ_LAT + 5/2 00001Reserved 01101READ_LAT + 11/4 00010READ_LAT 01110READ_LAT + 3 00011READ_LAT + 1/4 01111READ_LAT + 13/4 00100READ_LAT + 1/2 10000READ_LAT + 7/2 00101READ_LAT + 3/4 10001READ_LAT + 15/4 00110READ_LAT + 1 10010READ_LAT + 4 00111READ_LAT + 5/4 10011READ_LAT + 17/4 01000READ_LAT + 3/2 10100READ_LAT + 9/2 01001READ_LAT + 7/4 10101READ_LAT + 19/4 01010READ_LAT + 2 10110–11111 Reserved 01011READ_LAT + 9/4
9	—	Reserved
10–12	WR_LAT	Write latency. Note that the total write latency for DDR2 is equal to WR_LAT + ADD_LAT. 000 Reserved 001 1 clock 010 2 clocks 011 3 clocks 100 4 clocks 101 5 clocks 110 6 clocks 111 7 clocks
13–15	—	Reserved

Table 10-11. TIMING_CFG_2 Field Descriptions (continued)

Bits	Name	Description
16–18	RD_TO_PRE	Read to precharge (t_{RTP}). For DDR2, with a non-zero ADD_LAT value, takes a minimum of ADD_LAT + t_{RTP} cycles between read and precharge. 000 Reserved 100 4 cycles 001 1 cycle 101–111 Reserved 010 2 cycles 011 3 cycles
19–21	WR_DATA_DELAY	Write command to write data strobe timing adjustment. Controls the amount of delay applied to the data and data strobes for writes. See Section 10.5.7, “DDR SDRAM Write Timing Adjustments,” for details. 000 0 clock delay 100 1 clock delay 001 1/4 clock delay 101 5/4 clock delay 010 1/2 clock delay 110 3/2 clock delay 011 3/4 clock delay 111 Reserved
22	—	Reserved
23–25	CKE_PLS	Minimum CKE pulse width (t_{CKE}). 000 Reserved 011 3 cycles 001 1 cycle 100 4 cycles 010 2 cycles 101–111 Reserved
26–31	FOUR_ACT	Window for four activates (t_{FAW}). This is applied to DDR2 with eight logical banks only. 000000 Reserved ... 0000011 cycle 010011 19 cycles 0000102 cycles 01010020 cycles 0000113 cycles 010101–111111 Reserved 0001004 cycles

10.4.1.7 DDR SDRAM Control Configuration (DDR_SDRAM_CFG)

The DDR SDRAM control configuration register, shown in Figure 10-8, enables the interface logic and specifies certain operating features such as self refreshing, registered DIMMs, and dynamic power management.

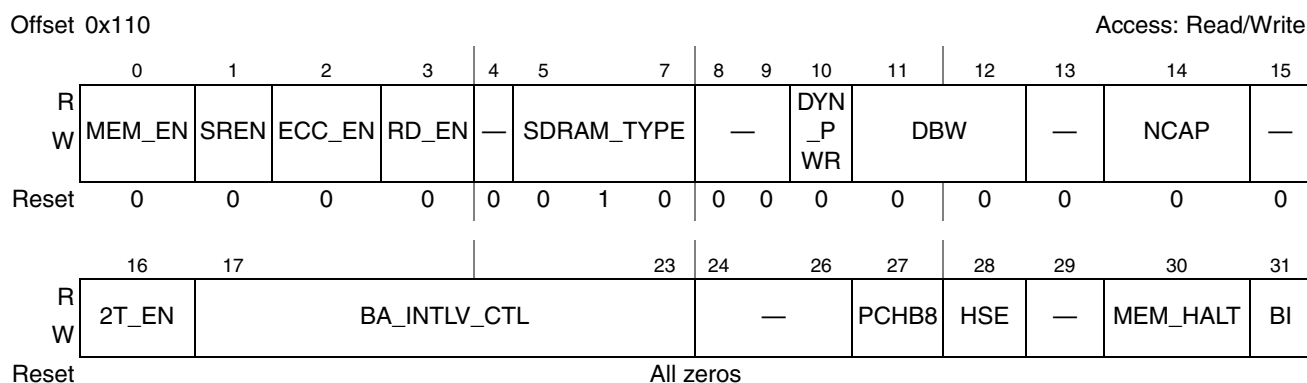


Figure 10-8. DDR SDRAM Control Configuration Register (DDR_SDRAM_CFG)

Table 10-12 describes the DDR_SDRAM_CFG fields.

Table 10-12. DDR_SDRAM_CFG Field Descriptions

Bits	Name	Description
0	MEM_EN	DDR SDRAM interface logic enable. 0 SDRAM interface logic is disabled. 1 SDRAM interface logic is enabled. Must not be set until all other memory configuration parameters have been appropriately configured by initialization code.
1	SREN	Self refresh enable (during sleep). 0 SDRAM self refresh is disabled during sleep. Whenever self-refresh is disabled, the system is responsible for preserving the integrity of SDRAM during sleep. 1 SDRAM self refresh is enabled during sleep.
2	ECC_EN	ECC enable. Note that uncorrectable read errors may cause an interrupt. 0 No ECC errors are reported. No ECC interrupts are generated. 1 ECC is enabled.
3	RD_EN	Registered DRAM module enable. Specifies the type of DRAM module used in the system. 0 Indicates unbuffered DRAM modules. 1 Indicates registered DRAM modules. Note: RD_EN and 2T_EN must not both be set at the same time.
4	—	Reserved
5–7	SDRAM_TYPE	Type of SDRAM device to be used. This field is used when issuing the automatic hardware initialization sequence to DRAM through Mode Register Set and Extended Mode Register Set commands. For DDR2 SDRAM, the field is set to 011.
8–9	—	Reserved
10	DYN_PWR	Dynamic power management mode 0 Dynamic power management mode is disabled. 1 Dynamic power management mode is enabled. If there is no ongoing memory activity, the SDRAM CKE signal is negated.
11–12	DBW	DRAM data bus width. 00 Reserved 01 32-bit bus is used 10 16-bit bus is used 11 Reserved
13	—	Reserved
14	NCAP	Non-concurrent auto-precharge. Some older DDR DRAMs do not support concurrent auto precharge. If one of these devices is used, then this bit needs to be set if auto precharge is used. 0 DRAMs in system support concurrent auto-precharge. 1 DRAMs in system do not support concurrent auto-precharge.
15	—	Reserved
16	2T_EN	Enable 2T timing. 0 1T timing is enabled. The DRAM command/address are held for only 1 cycle on the DRAM bus. 1 2T timing is enabled. The DRAM command/address are held for 2 full cycles on the DRAM bus for every DRAM transaction. However, the chip select is only held for the second cycle. Note: RD_EN and 2T_EN must not both be set at the same time.

Table 10-12. DDR_SDRAM_CFG Field Descriptions (continued)

Bits	Name	Description
17–23	BA_INTLV_CTL	Bank (chip select) interleaving control. Set this field only if you wish to use bank interleaving. (All unlisted field values are reserved.) 00000000 No external memory banks are interleaved 10000000 External memory banks 0 and 1 are interleaved
24–26	—	Reserved
27	PCHB8	Precharge bit 8 enable. 0 MA[10] is used to indicate the auto-precharge and precharge all commands. 1 MA[8] is used to indicate the auto-precharge and precharge all commands.
28	HSE	Global half-strength override Sets I/O driver impedance to half strength. This impedance is used by the address/command, data, and clock impedance values, but only if automatic hardware calibration is disabled and the corresponding group's software override is disabled in the DDR control driver register(s) described in Section 6.3.2.9, “DDR Control Driver Register (DDRCDR).” This bit should be cleared if using automatic hardware calibration. 0 I/O driver impedance is configured to full strength. 1 I/O driver impedance is configured to half strength.
29	—	Reserved
30	MEM_HALT	DDR memory controller halt. When this bit is set, the memory controller does not accept any new data read/write transactions to DDR SDRAM until the bit is cleared again. This can be used when bypassing initialization and forcing MODE REGISTER SET commands through software. 0 DDR controller accepts new transactions. 1 DDR controller finishes any remaining transactions, and then it remains halted until this bit is cleared by software.
31	BI	Bypass initialization 0 DDR controller cycles through initialization routine based on SDRAM_TYPE 1 Initialization routine is bypassed. Software is responsible for initializing memory through DDR_SDRAM_MODE2 register. If software is initializing memory, then the MEM_HALT bit can be set to prevent the DDR controller from issuing transactions during the initialization sequence. Note that the DDR controller does not issue a DLL reset to the DRAMs when bypassing the initialization routine, regardless of the value of DDR_SDRAM_CFG[DLL_RST_DIS]. If a DLL reset is required, then the controller should be forced to enter and exit self refresh after the controller is enabled.

10.4.1.8 DDR SDRAM Control Configuration 2 (DDR_SDRAM_CFG_2)

The DDR SDRAM control configuration register 2, shown in [Figure 10-9](#), provides more control configuration for the DDR controller.

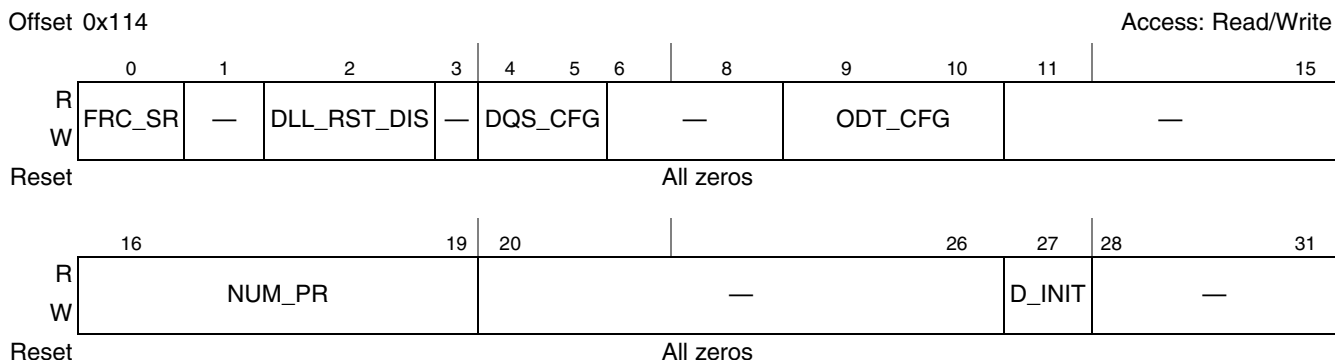


Figure 10-9. DDR SDRAM Control Configuration Register 2 (DDR_SDRAM_CFG_2)

[Table 10-13](#) describes the DDR_SDRAM_CFG_2 fields.

Table 10-13. DDR_SDRAM_CFG_2 Field Descriptions

Bits	Name	Description
0	FRC_SR	Force self refresh 0 DDR controller operates in normal mode. 1 DDR controller enters self-refresh mode.
1	—	Reserved. Should be cleared.
2	DLL_RST_DIS	DLL reset disable. The DDR controller typically issues a DLL reset to the DRAMs when exiting self refresh. However, this function may be disabled by setting this bit during initialization. 0 DDR controller issues a DLL reset to the DRAMs when exiting self refresh. 1 DDR controller does not issue a DLL reset to the DRAMs when exiting self refresh.
3	—	Reserved
4–5	DQS_CFG	DQS configuration 00 Only true DQS signals are used. 01 Reserved 10 Reserved 11 Reserved
6–8	—	Reserved
9–10	ODT_CFG	ODT configuration. This field defines how ODT is driven to the on-chip IOs. See Section 6.3.2.9, “DDR Control Driver Register (DDRCDR),” which defines the termination value that is used. 00 Never assert ODT to internal IOs 01 Assert ODT to internal IOs only during writes to DRAM 10 Assert ODT to internal IOs only during reads to DRAM 11 Always keep ODT asserted to internal IOs
11–15	—	Reserved.

Table 10-13. DDR_SDRAM_CFG_2 Field Descriptions (continued)

Bits	Name	Description
16–19	NUM_PR	Number of posted refreshes. This determines how many posted refreshes, if any, can be issued at one time. Note that if posted refreshes are used, then this field, along with DDR_SDRAM_INTERVAL[REFINT], must be programmed such that the maximum t_{ras} specification cannot be violated. 0000 Reserved 0001 1 refresh is issued at a time 0010 2 refreshes is issued at a time 0011 3 refreshes is issued at a time ... 1000 8 refreshes is issued at a time 1001–1111 Reserved
20–26	—	Reserved, should be cleared.
27	D_INIT	DRAM data initialization. This bit is set by software, and it is cleared by hardware. If software sets this bit before the memory controller is enabled, the controller automatically initializes DRAM after it is enabled. This bit is automatically cleared by hardware once the initialization is completed. This data initialization bit should only be set when the controller is idle. 0 There is not data initialization in progress, and no data initialization is scheduled 1 The memory controller initializes memory once it is enabled. This bit remains asserted until the initialization is complete. The value in DDR_DATA_INIT register is used to initialize memory.
28–31	—	Reserved

10.4.1.9 DDR SDRAM Mode Configuration (DDR_SDRAM_MODE)

The DDR SDRAM mode configuration register, shown in [Figure 10-10](#), sets the values loaded into the DDR’s mode registers.



Figure 10-10. DDR SDRAM Mode Configuration Register (DDR_SDRAM_MODE)

Table 10-14 describes the DDR_SDRAM_MODE fields.

Table 10-14. 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 10-10, 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 10-10, corresponds to SDMODE[15]. The msb of the SDRAM mode register value must be stored at SDMODE[0]. Because the memory controller forces SDMODE[7] to certain values depending on the state of the initialization sequence, (for resetting the SDRAM's DLL) the corresponding bits of this field are ignored by the memory controller. Note that SDMODE[7] is mapped to MA[8].

10.4.1.10 DDR SDRAM Mode 2 Configuration (DDR_SDRAM_MODE_2)

The DDR SDRAM mode 2 configuration register, shown in Figure 10-11, sets the values loaded into the DDR's extended mode 2 and 3 registers (for DDR2).



Figure 10-11. DDR SDRAM Mode 2 Configuration Register (DDR_SDRAM_MODE_2)

Table 10-15 describes the DDR_SDRAM_MODE_2 fields.

Table 10-15. 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 10-11, 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 10-11, corresponds to ESDMODE3[15]. The msb of the SDRAM extended mode 3 register value must be stored at ESDMODE3[0].

10.4.1.11 DDR SDRAM Mode Control Register (DDR_SDRAM_MD_CNTL)

The DDR SDRAM mode control register, shown in Figure 10-12, 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 10-16 describes the fields of this register. Table 10-17 shows the user how to set the fields of this register to accomplish the above tasks.

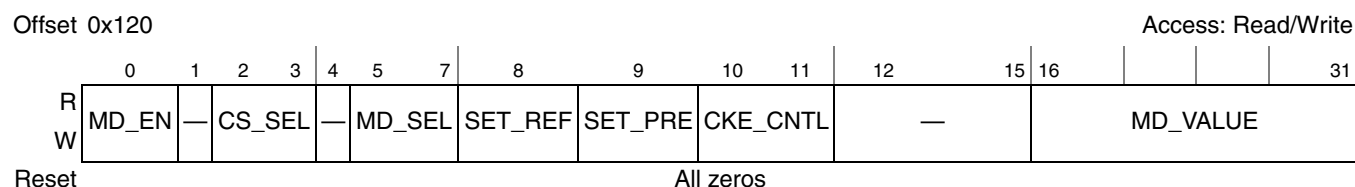


Figure 10-12. DDR SDRAM Mode Control Register (DDR_SDRAM_MD_CNTL)

Table 10-16 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 10-16. DDR_SDRAM_MD_CNTL Field Descriptions

Bits	Name	Description
0	MD_EN	Mode enable. Setting this bit specifies that valid data in MD_VALUE is ready to be written to DRAM as one of the following commands: <ul style="list-style-type: none"> • 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 is driven active due to any command forced by software in DDR_SDRAM_MD_CNTL. 00 Chip select 0 is active 01 Chip select 1 is active 10 Reserved 11 Reserved
4	—	Reserved

Table 10-16. DDR_SDRAM_MD_CNTL Field Descriptions (continued)

Bits	Name	Description
5–7	MD_SEL	<p>Mode register select. MD_SEL specifies one of the following:</p> <ul style="list-style-type: none"> 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. <p>Note that MD_SEL contains the value that is presented onto the memory bank address pins (MBA_n) of the DDR controller.</p> <p>000 MR 001 EMR 010 EMR2 011 EMR3</p>
8	SET_REF	<p>Set refresh. Forces an immediate refresh to be issued to the chip select specified by DDR_SDRAM_MD_CNTL[CS_SEL]. This bit is set by software and cleared by hardware once the command has been issued.</p> <p>0 Indicates that no refresh command needs to be issued. 1 Indicates that a refresh command is ready to be issued.</p>
9	SET_PRE	<p>Set precharge. Forces a precharge or precharge all to be issued to the chip select specified by DDR_SDRAM_MD_CNTL[CS_SEL]. This bit is set by software and cleared by hardware once the command has been issued.</p> <p>0 Indicates that no precharge all command needs to be issued. 1 Indicates that a precharge all command is ready to be issued.</p>
10–11	CKE_CNTL	<p>Clock enable control. Allows software to globally clear or set all CKE signals issued to DRAM. Once software has forced the value driven on CKE, that value continues to be forced until software clears the CKE_CNTL bits. At that time, the DDR controller continues to drive the CKE signals to the same value forced by software until another event causes the CKE signals to change (such as, self refresh entry/exit, power down entry/exit).</p> <p>00 CKE signals are not forced by software. 01 CKE signals are forced to a low value by software. 10 CKE signals are forced to a high value by software. 11 Reserved</p>
12–15	—	Reserved
16–31	MD_VALUE	<p>Mode register value. This field, which specifies the value that is presented on the memory address pins of the DDR controller during a mode register set command, is significant only when this register is used to issue a mode register set command or a precharge or precharge all command.</p> <p>For a mode register set command, this field contains the data to be written to the selected mode register. For a precharge command, only bit five is significant:</p> <p>0 Issue a precharge command; MD_SEL selects the logical bank to be precharged 1 Issue a precharge all command; all logical banks are precharged</p>

Table 10-17 shows how DDR_SDRAM_MD_CNTL fields should be set for each of the tasks described above.

Table 10-17. Settings of DDR_SDRAM_MD_CNTL Fields

Field	Mode Register Set	Refresh	Precharge	Clock Enable Signals Control
MD_EN	1	0	0	—
SET_REF	0	1	0	—
SET_PRE	0	0	1	—

Table 10-17. Settings of DDR_SDRAM_MD_CNTL Fields (continued)

Field	Mode Register Set	Refresh	Precharge	Clock Enable Signals Control
CS_SEL	Chooses chip select (CS)			—
MD_SEL	Select mode register. See Table 10-16 .	—	Selects logical bank	—
MD_VALUE	Value written to mode register	—	Only bit five is significant. See Table 10-16 .	—
CKE_CNTL	0	0	0	See Table 10-16 .

10.4.1.12 DDR SDRAM Interval Configuration (DDR_SDRAM_INTERVAL)

The DDR SDRAM interval configuration register, shown in [Figure 10-13](#), 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 10-13. DDR SDRAM Interval Configuration Register (DDR_SDRAM_INTERVAL)

[Table 10-18](#) describes the DDR_SDRAM_INTERVAL fields.

Table 10-18. DDR_SDRAM_INTERVAL Field Descriptions

Bits	Name	Description
0–15	REFINT	Refresh interval. Represents the number of memory bus clock cycles between refresh cycles. Depending on DDR_SDRAM_CFG_2[NUM_PR], some number of rows are refreshed in each DDR SDRAM physical bank during each refresh cycle. The value for REFINT depends on the specific SDRAMs used and the interface clock frequency. Refreshes are not issued when the REFINT is set to all 0s.
16–17	—	Reserved
18–31	BSTOPRE	Precharge interval. Sets the duration (in memory bus clocks) that a page is retained after a DDR SDRAM access. If BSTOPRE is zero, the DDR memory controller uses auto-precharge read and write commands rather than operating in page mode. This is called global auto-precharge mode.

10.4.1.13 DDR SDRAM Data Initialization (DDR_DATA_INIT)

The DDR SDRAM data initialization register, shown in [Figure 10-14](#), provides the value that is used to initialize memory if DDR_SDRAM_CFG2[D_INIT] is set.

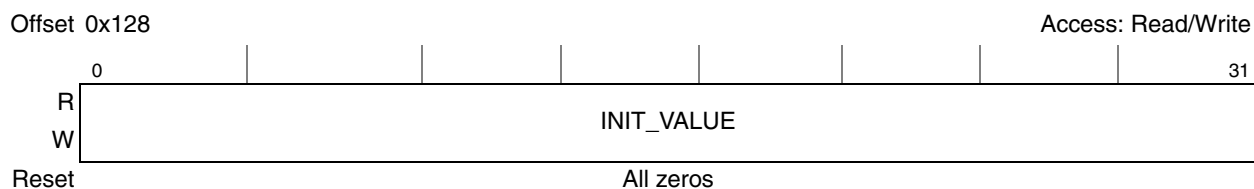


Figure 10-14. DDR SDRAM Data Initialization Configuration Register (DDR_DATA_INIT)

[Table 10-19](#) describes the DDR_DATA_INIT fields.

Table 10-19. DDR_DATA_INIT Field Descriptions

Bits	Name	Description
0–31	INIT_VALUE	Initialization value. Represents the value that DRAM is initialized with if DDR_SDRAM_CFG2[D_INIT] is set.

10.4.1.14 DDR SDRAM Clock Control (DDR_SDRAM_CLK_CNTL)

The DDR SDRAM clock control configuration register, shown in [Figure 10-15](#), provides a 1/4-cycle clock adjustment.

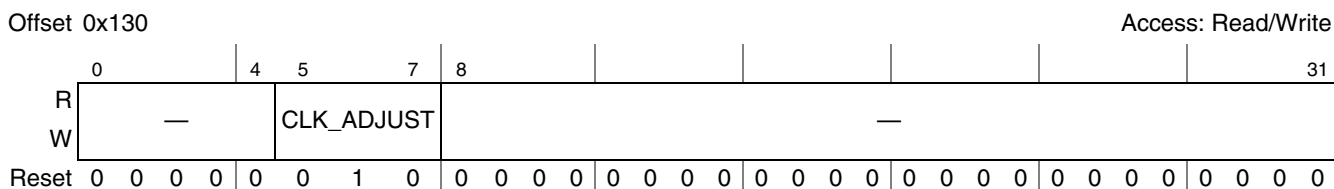


Figure 10-15. DDR SDRAM Clock Control Configuration Register (DDR_SDRAM_CLK_CNTL)

[Table 10-20](#) describes the DDR_SDRAM_CLK_CNTL fields.

Table 10-20. DDR_SDRAM_CLK_CNTL Field Descriptions

Bits	Name	Description
0–4	—	Reserved
5–7	CLK_ADJUST	Clock adjust. 000 Clock is launched aligned with address/command 001 Clock is launched 1/4 applied cycle after address/command 010 Clock is launched 1/2 applied cycle after address/command 011 Clock is launched 3/4 applied cycle after address/command 100 Clock is launched 1 applied cycle after address/command 101–111 Reserved
8	—	Reserved, should be cleared.
9–31	—	Reserved

10.4.1.15 DDR Initialization Address (DDR_INIT_ADDR)

The DDR SDRAM initialization address register, shown in Figure 10-16, provides the address that is used for the automatic CAS to preamble calibration after POR.

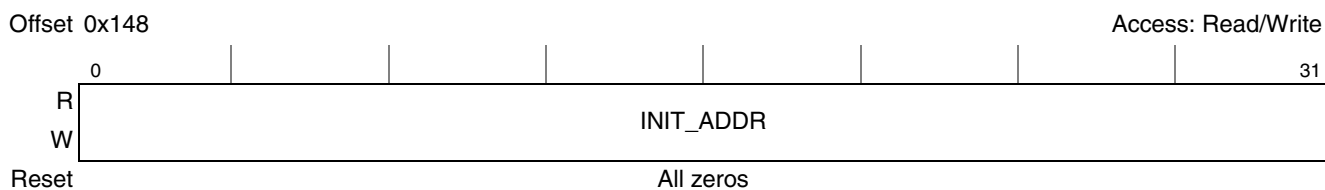


Figure 10-16. DDR Initialization Address Configuration Register (DDR_INIT_ADDR)

Table 10-21 describes the DDR_INIT_ADDR fields.

Table 10-21. DDR_INIT_ADDR Field Descriptions

Bits	Name	Description
0–31	INIT_ADDR	Initialization address. Represents the address that is used for the automatic CAS to preamble calibration at POR.

10.4.1.16 DDR IP Block Revision 1 (DDR_IP_REV1)

The DDR IP block revision 1 register, shown in Figure 10-17, provides read-only fields with the IP block ID, along with major and minor revision information.

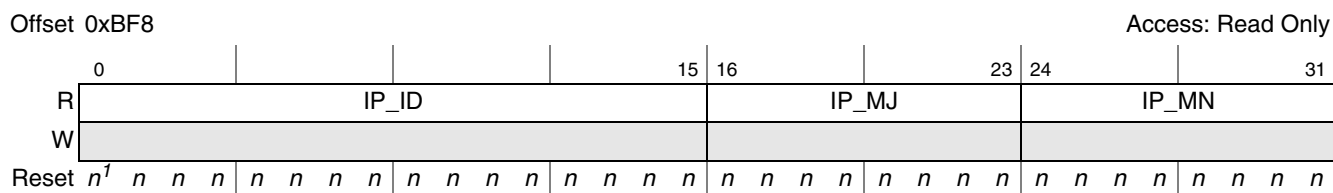


Figure 10-17. DDR IP Block Revision 1 (DDR_IP_REV1)

¹ For reset values, see Table 10-22.

Table 10-22 describes the DDR_IP_REV1 fields.

Table 10-22. DDR_IP_REV1 Field Descriptions

Bits	Name	Description
0–15	IP_ID	IP block ID. For the DDR controller, this value is 0x0002.
16–23	IP_MJ	Major revision. This is currently set to 0x02.
24–31	IP_MN	Minor revision. This is currently set to 0x01.

10.4.1.17 DDR IP Block Revision 2 (DDR_IP_REV2)

The DDR IP block revision 2 register, shown in [Figure 10-18](#), provides read-only fields with the IP block integration and configuration options.

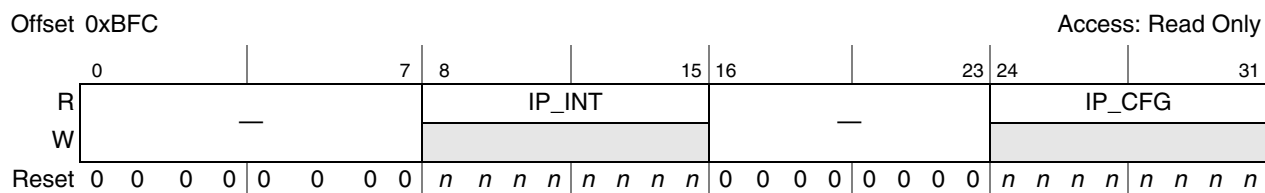


Figure 10-18. DDR IP Block Revision 2 (DDR_IP_REV2)

[Table 10-23](#) describes the DDR_IP_REV2 fields.

Table 10-23. DDR_IP_REV2 Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8–15	IP_INT	IP block integration options. This is currently set to 0x0000.
16–23	—	Reserved
24–31	IP_CFG	IP block configuration options. This is currently set to 0x82.

10.4.1.18 Memory Data Path Error Injection Mask High (DATA_ERR_INJECT_HI)

The memory data path error injection mask high register is shown in [Figure 10-19](#).

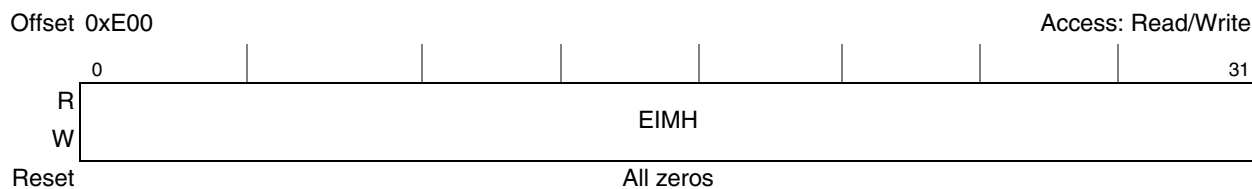


Figure 10-19.

[Table 10-24](#) describes the DATA_ERR_INJECT_HI fields.

Table 10-24.

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.

10.4.1.19 Memory Data Path Error Injection Mask Low (DATA_ERR_INJECT_LO)

The memory data path error injection mask low register is shown in [Figure 10-20](#).

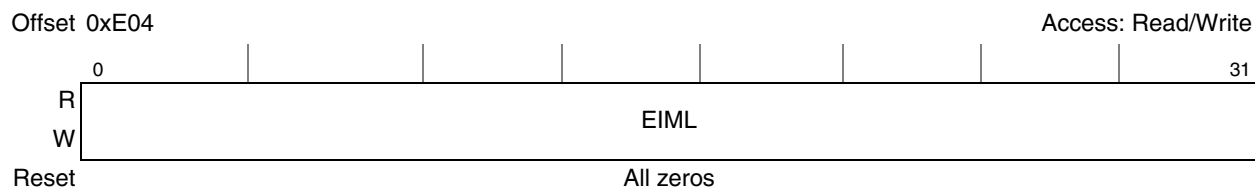


Figure 10-20. Memory Data Path Error Injection Mask Low Register (DATA_ERR_INJECT_LO)

[Table 10-25](#) describes the DATA_ERR_INJECT_LO fields.

Table 10-25. 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.

10.4.1.20 Memory Data Path Error Injection Mask ECC (ERR_INJECT)

The memory data path error injection mask ECC register, shown in [Figure 10-21](#), sets the ECC mask, enables errors to be written to ECC memory, and allows the ECC byte to mirror the most significant data byte.

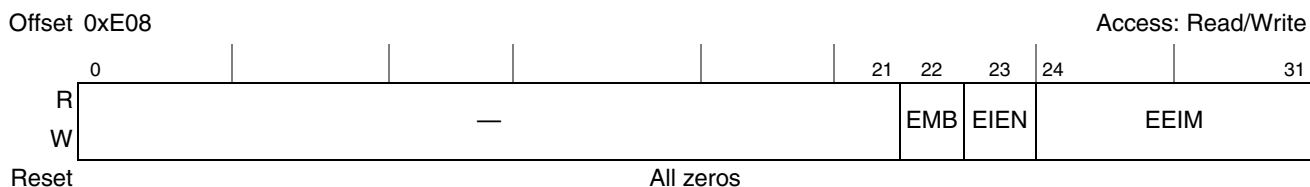


Figure 10-21. Memory Data Path Error Injection Mask ECC Register (ERR_INJECT)

[Table 10-26](#) describes the ERR_INJECT fields.

Table 10-26. ERR_INJECT Field Descriptions

Bits	Name	Description
0–21	—	Reserved
22	EMB	ECC mirror byte 0 Mirror byte functionality disabled. 1 Mirror the most significant data path byte onto the ECC byte.
23	EIEN	Error injection enable 0 Error injection disabled. 1 Error injection enabled. This applies to the data mask bits, the ECC mask bits, and the ECC mirror bit. Note that error injection should not be enabled until the memory controller has been enabled through DDR_SDRAM_CFG[MEM_EN].
24–31	EEIM	ECC error injection mask. Setting a mask bit causes the corresponding ECC bit to be inverted on memory bus writes.

10.4.1.21 Memory Data Path Read Capture High (CAPTURE_DATA_HI)

The memory data path read capture high register, shown in [Figure 10-22](#), stores the high word of the read data path during error capture.

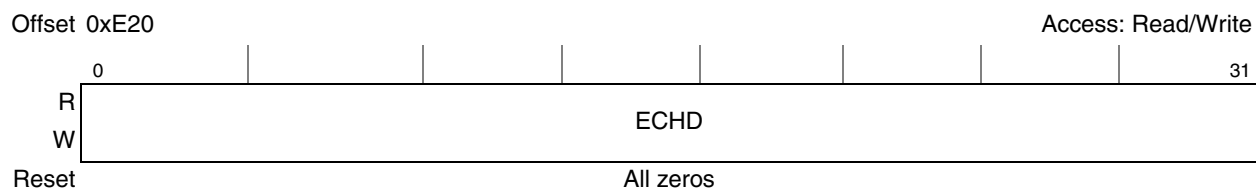


Figure 10-22. Memory Data Path Read Capture High Register (CAPTURE_DATA_HI)

[Table 10-27](#) describes the CAPTURE_DATA_HI fields.

Table 10-27. 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.

10.4.1.22 Memory Data Path Read Capture Low (CAPTURE_DATA_LO)

The memory data path read capture low register, shown in [Figure 10-23](#), stores the low word of the read data path during error capture.

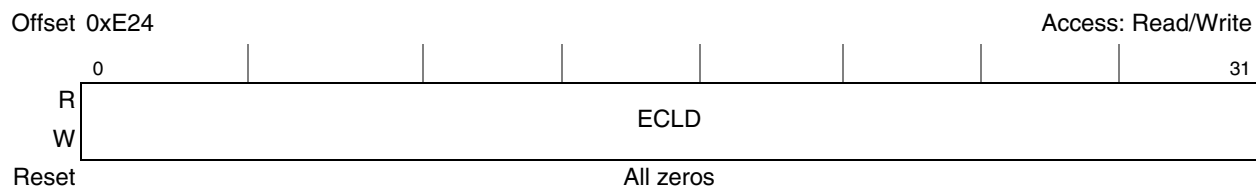


Figure 10-23. Memory Data Path Read Capture Low Register (CAPTURE_DATA_LO)

[Table 10-28](#) describes the CAPTURE_DATA_LO fields.

Table 10-28. 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.

10.4.1.23 Memory Data Path Read Capture ECC (CAPTURE_ECC)

The memory data path read capture ECC register, shown in [Figure 10-24](#), stores the ECC syndrome bits that were on the data bus when an error was detected.

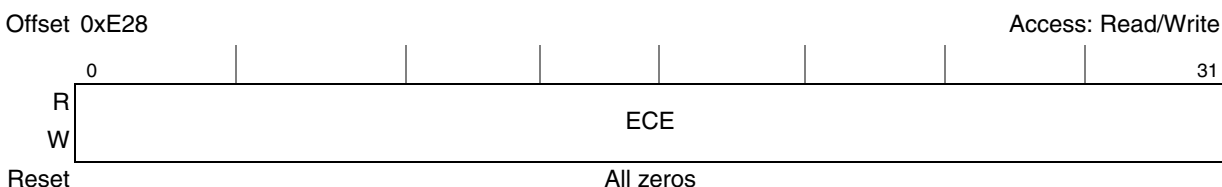


Figure 10-24. Memory Data Path Read Capture ECC Register (CAPTURE_ECC)

[Table 10-29](#) describes the CAPTURE_ECC fields.

Table 10-29. CAPTURE_ECC Field Descriptions

Bits	Name	Description
0–31	ECC	Reserved <ul style="list-style-type: none"> 0–7: 8-bit ECC for first 16 bits in 16-bit bus mode; should be ignored for 32-bit 8–15: 8-bit ECC for second 16 bits in 16-bit bus mode; 1st 32 bits in 32-bit bus mode 16–23: 8-bit ECC for third 16 bits in 16-bit bus mode; should be ignored for 32-bit 24–31: 8-bit ECC for fourth 16 bits in 16-bit bus mode; 2nd 32 bits in 32-bit bus mode

10.4.1.24 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 10-25](#).

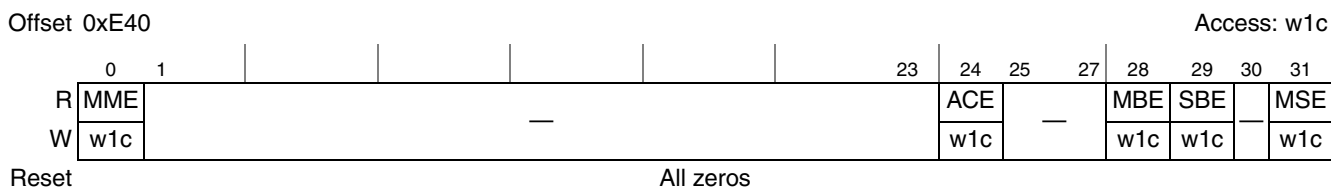


Figure 10-25. Memory Error Detect Register (ERR_DETECT)

[Table 10-30](#) describes the ERR_DETECT fields.

Table 10-30. ERR_DETECT Field Descriptions

Bits	Name	Description
0	MME	Multiple memory errors. This bit is cleared by software writing a 1. 0 Multiple memory errors of the same type were not detected. 1 Multiple memory errors of the same type were detected.
1–23	—	Reserved

Table 10-31. ERR_DISABLE Field Descriptions (continued)

Bits	Name	Description
30	—	Reserved
31	MSED	Memory select error disable 0 Memory select errors are enabled. 1 Memory select errors are disabled.

10.4.1.26 Memory Error Interrupt Enable (ERR_INT_EN)

The memory error interrupt enable register, shown in [Figure 10-27](#), 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).

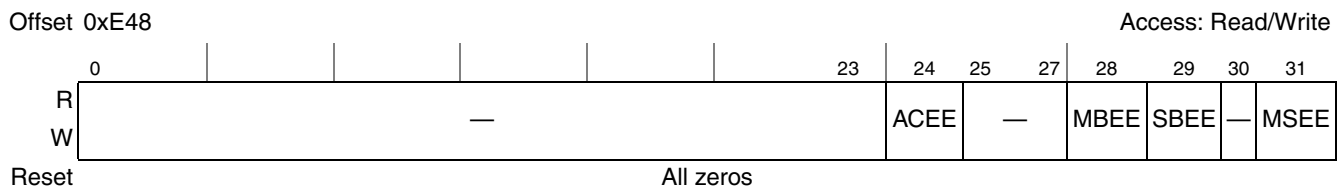


Figure 10-27. Memory Error Interrupt Enable Register (ERR_INT_EN)

[Table 10-32](#) describes the ERR_INT_EN fields.

Table 10-32. ERR_INT_EN Field Descriptions

Bits	Name	Description
0-23	—	Reserved
24	ACEE	Automatic calibration error interrupt enable 0 Automatic calibration errors cannot generate interrupts. 1 Automatic calibration errors generate interrupts.
25-27	—	Reserved
28	MBEE	Multiple-bit ECC error interrupt enable. 0 Multiple-bit ECC errors cannot generate interrupts. 1 Multiple-bit ECC errors generate interrupts.
29	SBEE	Single-bit ECC error interrupt enable 0 Single-bit ECC errors cannot generate interrupts. 1 Single-bit ECC errors generate interrupts.
30	—	Reserved
31	MSEE	Memory select error interrupt enable 0 Memory select errors do not cause interrupts. 1 Memory select errors generate interrupts.

10.4.1.27 Memory Error Attributes Capture (CAPTURE_ATTRIBUTES)

The memory error attributes capture register, shown in [Figure 10-28](#), sets attributes for errors including type, size, source, and others.

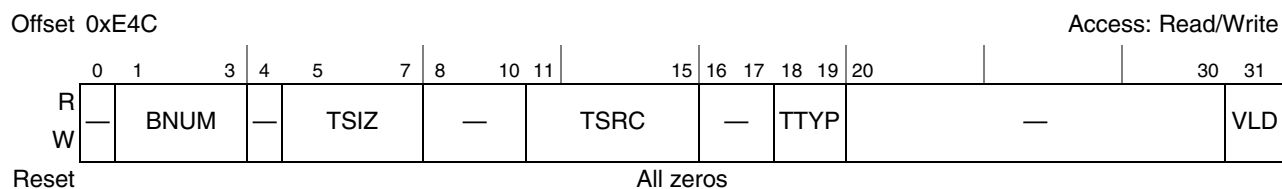


Figure 10-28. Memory Error Attributes Capture Register (CAPTURE_ATTRIBUTES)

[Table 10-33](#) describes the CAPTURE_ATTRIBUTES fields.

Table 10-33. CAPTURE_ATTRIBUTES Field Descriptions

Bits	Name	Description
0	—	Reserved
1–3	BNUM	Data beat number. Captures the doubleword number for the detected error. Relevant only for ECC errors.
4	—	Reserved
5–7	TSIZ	Transaction size for the error. Captures the transaction size in double words. 000 4 double words 001 1 double word 010 2 double words 011 3 double words Others Reserved
8–10	—	Reserved
11–15	TSRC	Transaction source for the error 00000 e300 core data transaction 00001 Reserved 00010 e300 core instruction fetch 00011 Reserved 00100 eTSEC1 00101–01000 Reserved 01001 I ² C (boot sequencer) 01010 JTAG 01011 Reserved 01100 eSDHC 01101–11100 Reserved 11101 PCI Express 11110 Reserved 11111 DMA
16–17	—	Reserved
18–19	TTYP	Transaction type for the error. 00 Reserved 01 Write 10 Read 11 Read-modify-write
20–30	—	Reserved
31	VLD	Valid. Set as soon as valid information is captured in the error capture registers.

10.4.1.28 Memory Error Address Capture (CAPTURE_ADDRESS)

The memory error address capture register, shown in [Figure 10-29](#), holds the 32 lsbs of a transaction when a DDR ECC error is detected.

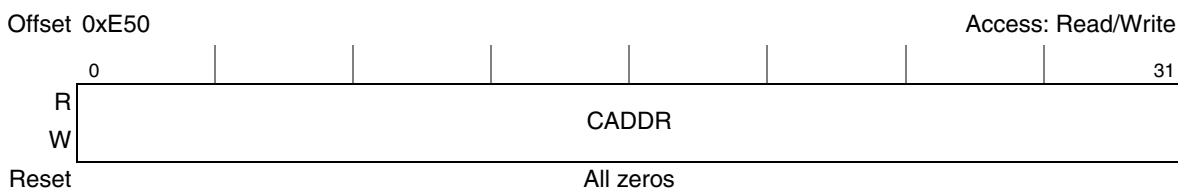


Figure 10-29. Memory Error Address Capture Register (CAPTURE_ADDRESS)

[Table 10-34](#) describes the CAPTURE_ADDRESS fields.

Table 10-34. 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.

10.4.1.29 Single-Bit ECC Memory Error Management (ERR_SBE)

The single-bit ECC memory error management register, shown in [Figure 10-30](#), 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.

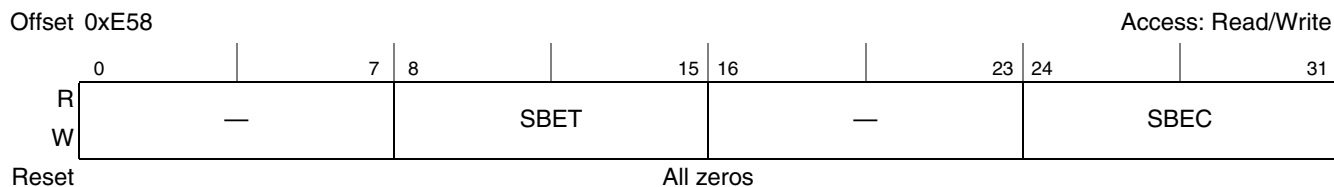


Figure 10-30. Single-Bit ECC Memory Error Management Register (ERR_SBE)

[Table 10-35](#) describes the ERR_SBE fields.

Table 10-35. ERR_SBE Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8–15	SBET	Single-bit error threshold. Establishes the number of single-bit errors that must be detected before an error condition is reported. As a special case, setting this to zero means error threshold is set to 256.
16–23	—	Reserved
24–31	SBEC	Single-bit error counter. Indicates the number of single-bit errors detected and corrected since the last error report. If single-bit error reporting is enabled, an error is reported and an interrupt is generated when this value equals SBET. SBEC is automatically cleared when the threshold value is reached.

10.5 Functional Description

The DDR SDRAM controller controls processor and I/O interactions with system memory. It provides support for JEDEC-compliant DDR2 SDRAM. The memory system allows a wide range of memory devices to be mapped to any arbitrary chip select, and support is provided for registered DIMMs and unbuffered DIMMs. However, registered DIMMs cannot be mixed with unbuffered DIMMs.

Figure 10-1 is a high-level block diagram of the DDR memory controller. Requests are received from the internal mastering device and the address is decoded to generate the physical bank, logical bank, row, and column addresses. The transaction is compared with values in the row open table to determine if the address maps to an open page. If the transaction does not map to an open page, an active command is issued.

The memory interface supports as many as two physical banks of 16-/24-/32-/40-bit wide memory. Bank sizes up to 512 Mbytes are supported, providing up to a maximum of 1 Gbytes of DDR main memory.

Programmable parameters allow for a variety of memory organizations and timings. Optional error checking and correcting (ECC) protection is provided for the DDR SDRAM data bus. Using ECC, the DDR memory controller detects and corrects all single-bit errors within the 32-bit data bus, detects all double-bit errors within the 32-bit data bus, and detects all errors within a nibble. The controller allows as many as 16 pages to be open simultaneously. The amount of time (in clock cycles) the pages remain open is programmable with `DDR_SDRAM_INTERVAL[BSTOPRE]`.

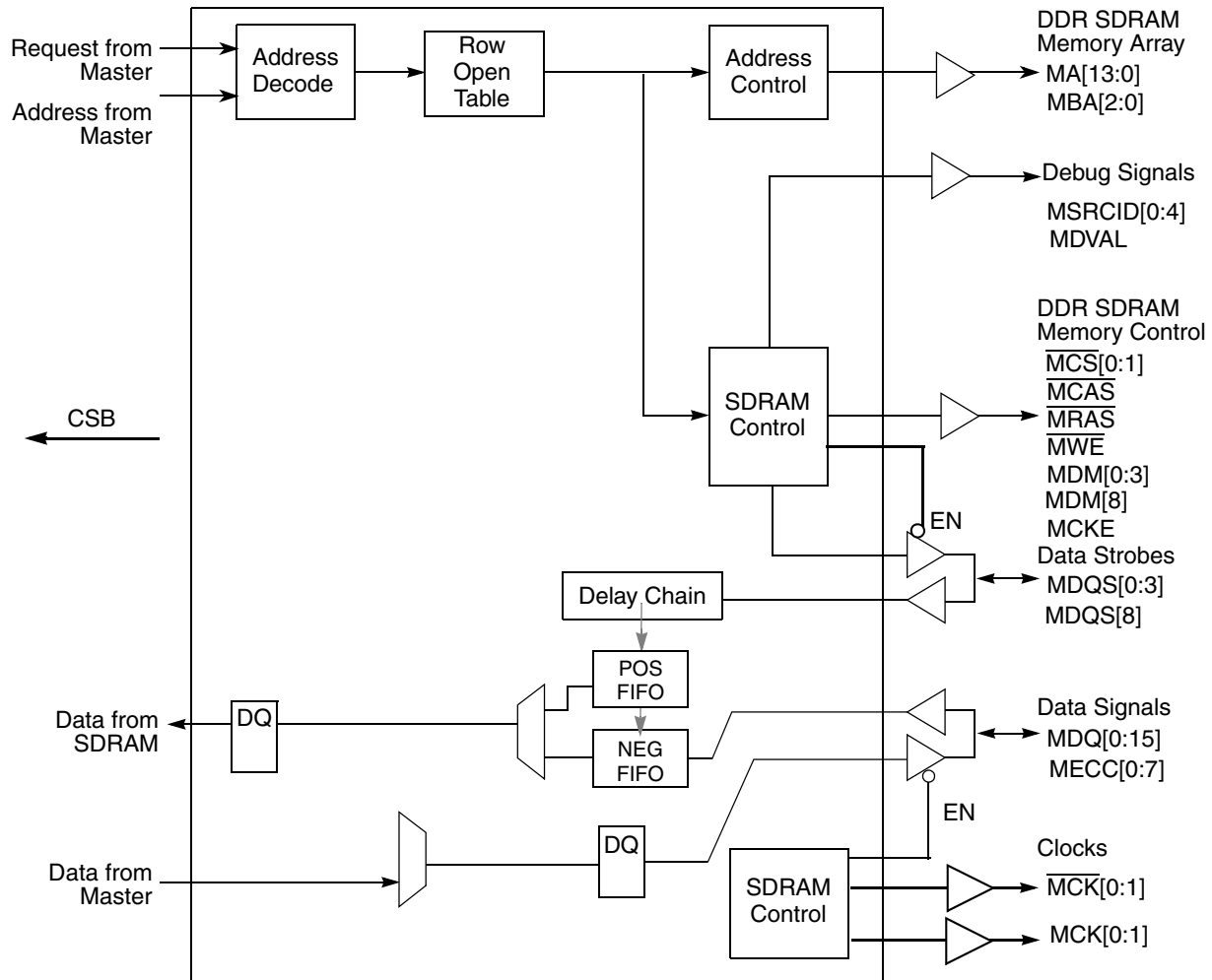


Figure 10-31. DDR Memory Controller Block Diagram

Read and write accesses to memory are burst oriented; accesses start at a selected location and continue for a programmed number of higher locations (4) 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[0:3] and MDQS[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 10-32 shows an example DDR SDRAM configuration with four logical banks.

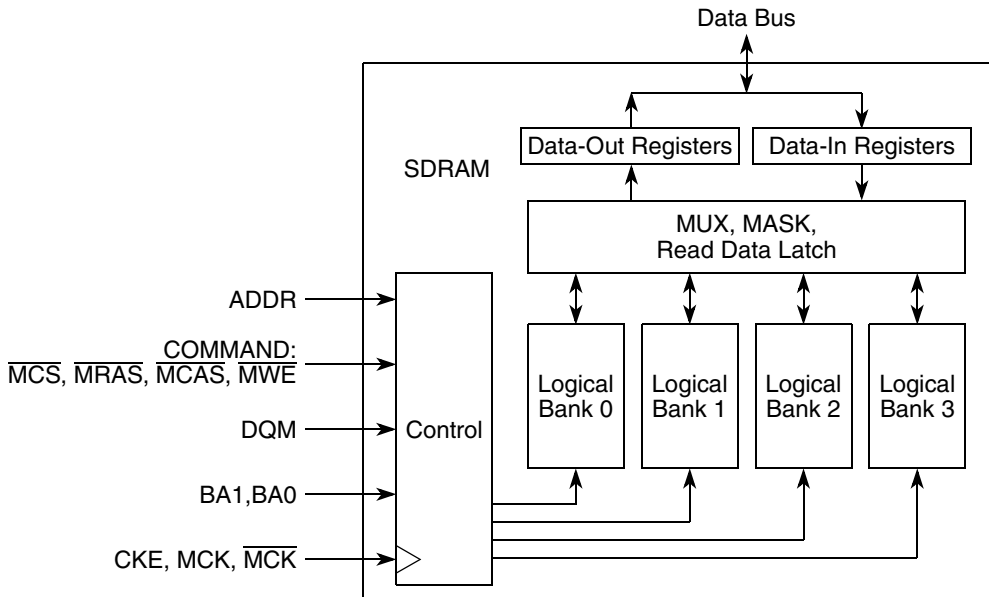


Figure 10-32. Typical Dual Data Rate SDRAM Internal Organization

Figure 10-33 shows some typical signal connections.

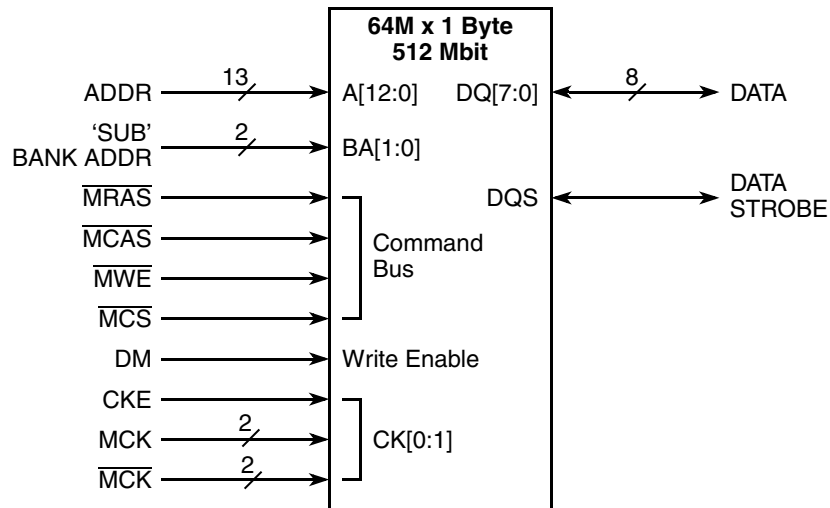
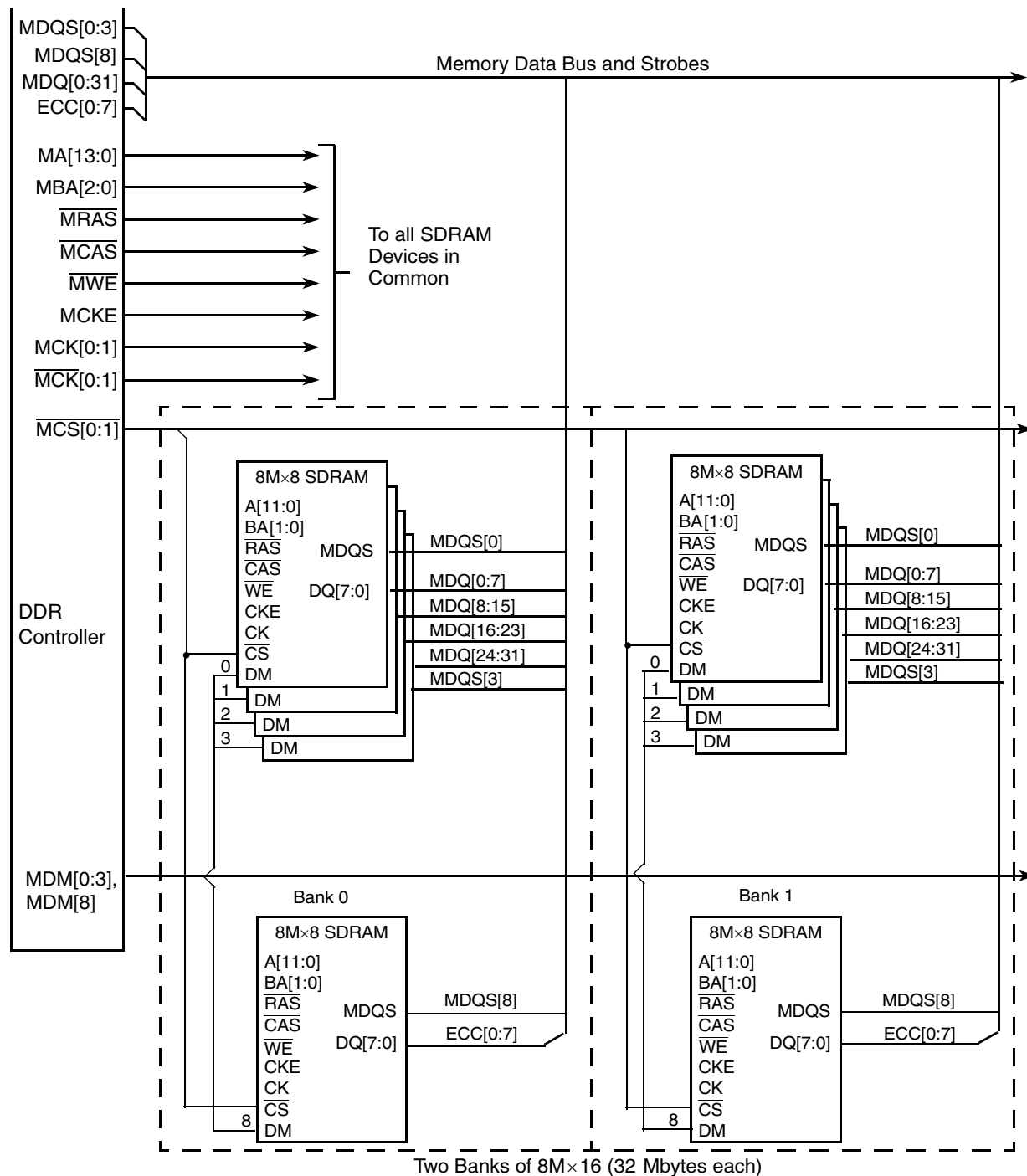


Figure 10-33. Typical DDR SDRAM Interface Signals

Figure 10-34 shows an example DDR SDRAM configuration with two physical banks each comprised of four $8M \times 8$ DDR modules for a total of 64 Mbytes of system memory. One of the nine modules is used for the memory's ECC checking function. Certain address and control lines may require buffering. Analysis of the device's AC timing specifications, desired memory operating frequency, capacitive loads,

and board routing loads can assist the system designer in deciding signal buffering requirements. The DDR memory controller drives 14 address pins, but in this example the DDR SDRAM devices use only 12 bits.



Two Banks of 8M×16 (32 Mbytes each)

1. All signals are connected in common (in parallel) except for MCS[0:1], MCK[0:1], MCK[0:1], MDM[0:3], MDM[8], and the data bus signals.
2. Each of the MCS[0:1] signals correspond with a separate physical bank of memory.
3. Buffering may be needed if large memory arrays are used.

Figure 10-34. Example 64-Mbyte DDR SDRAM Configuration With ECC

For information on how the DDR2 memory controller handles errors, see [Section 10.5.12, “Error Management.”](#)

10.5.1 DDR SDRAM Interface Operation

The DDR memory controller supports many different DDR SDRAM configurations. SDRAMs with different sizes can be used in the same system. Fourteen multiplexed address signals and three logical bank select signals support device densities from 64 Mbits to 2 Gbits. Two chip select (\overline{CS}) signals support two banks of DIMM of memory. The DDR SDRAM physical banks can be built from standard memory modules or directly-attached memory devices. The data path to individual physical banks is 32 bits wide, 40 bits with ECC. The DDR memory controller supports physical bank sizes from 16 Mbytes to 512 Mbytes. The physical banks can be constructed using $\times 8$, $\times 16$, or $\times 32$ memory devices. The memory technologies supported are 64 Mbits, 128 Mbits, 256 Mbits, 512 Mbits, and 1 Gbit. Some 2-Gbit devices are supported depending on the internal device configuration. Four 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 10-36](#) shows the DDR memory controller’s relationships between data byte lane0–3, MDM[0:3], MDQS[0:3], and MDQ[0:31] when DDR SDRAM memories are used with $\times 8$ or $\times 16$ devices.

Table 10-36. Byte Lane to Data Relationship

Data Byte Lane	Data Bus Mask	Data Bus Strobe	Data Bus
0 (MSB)	MDM[0]	MDQS[0]	MDQ[0:7]
1	MDM[1]	MDQS[1]	MDQ[8:15]
2	MDM[2]	MDQS[2]	MDQ[16:23]
3	MDM[3]	MDQS[3]	MDQ[24:31]

10.5.1.1 Supported DDR SDRAM Organizations

Although the DDR memory controller multiplexes row and column address bits onto 14 memory address signals and 3 logical bank select signals, a physical bank may be implemented with memory devices requiring fewer than 30 address bits. The physical bank may be configured to provide from 12 to 14 row address bits, plus 2 to 3 logical bank-select bits and from 8–11 column address bits.

[Table 10-37](#) describe DDR SDRAM device configurations supported by the DDR memory controller.

NOTE

DDR SDRAM is limited to 30 total address bits.

Table 10-37. Supported DDR2 SDRAM Device Configurations

SDRAM Device	Device Configuration	Row × Column × Bank Bits	32-Bit Bank Size	Two Banks of Memory
256 Mbits	32 Mbits × 8	13 × 10 × 2	128 Mbytes	256 Mbytes
256 Mbits	16 Mbits × 16	13 × 9 × 2	64 Mbytes	128 Mbytes
512 Mbits	64 Mbits × 8	14 × 10 × 2	256 Mbytes	512 Mbytes
512 Mbits	32 Mbits × 16	13 × 10 × 2	128 Mbytes	256 Mbytes
1 Gbits	128 Mbits × 8	14 × 10 × 3	512 Mbytes	1 Gbyte
1 Gbits	64 Mbits × 16	13 × 10 × 3	256 Mbytes	512 Mbytes
2 Gbits	128 Mbits × 16	14 × 10 × 3	512 Mbytes	1 Gbytes

If a transaction request is issued to the DDR memory controller and the address does not lie within any of the programmed address ranges for an enabled chip select, a memory select error is flagged. Errors are described in detail in [Section 10.5.12, “Error Management.”](#)

Using a memory-polling algorithm at power-on reset or by querying the JEDEC serial presence detect capability of memory modules, system firmware uses the memory-boundary registers to configure the DDR memory controller to map the size of each bank in memory. The memory controller uses its bank map to assert the appropriate \overline{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.

10.5.2 DDR SDRAM Address Multiplexing

The following tables ([Table 10-38](#) and [Table 10-39](#)) show the address bit encodings for each DDR SDRAM configuration. The address presented at the memory controller signals MA[13:0] use MA[13] as the msb and MA[0] as the lsb. Also, MA[10] is used as the auto-precharge bit in DDR2 modes for reads and writes, so the column address can never use MA[10].

Table 10-38. DDR2 Address Multiplexing for 32-Bit Data Bus with Interleaving and Partial Array Self Refresh Disabled

Row × Col	msb	Address from Core Master																												lsb				
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27		28	29	30–31	
14 × 10 × 3	\overline{MRAS}				13	12	11	10	9	8	7	6	5	4	3	2	1	0																
	\overline{MBA}																		2	1	0													
	\overline{MCAS}																						9	8	7	6	5	4	3	2	1	0		
14 × 10 × 2	\overline{MRAS}				13	12	11	10	9	8	7	6	5	4	3	2	1	0																
	\overline{MBA}																			1	0													
	\overline{MCAS}																						9	8	7	6	5	4	3	2	1	0		
13 × 10 × 3	\overline{MRAS}				12	11	10	9	8	7	6	5	4	3	2	1	0																	
	\overline{MBA}																			2	1	0												
	\overline{MCAS}																						9	8	7	6	5	4	3	2	1	0		

Table 10-38. DDR2 Address Multiplexing for 32-Bit Data Bus with Interleaving and Partial Array Self Refresh Disabled

Row × Col	msb	Address from Core Master																													lsb	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30-31	
13 × 10 × 2	MRAS					12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																		1	0												
	MCAS																					9	8	7	6	5	4	3	2	1	0	
13 × 9 × 2	MRAS					12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																		1	0												
	MCAS																					8	7	6	5	4	3	2	1	0		

Table 10-39. DDR2 Address Multiplexing for 16-Bit Data Bus (with Interleaving Disabled)

Row × Col	msb	Address from Core Master																														lsb	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
14 × 10 × 3	MRAS					13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																		2	1	0												
	MCAS																					9	8	7	6	5	4	3	2	1	0		
14 × 10 × 2	MRAS					13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																		1	0													
	MCAS																					9	8	7	6	5	4	3	2	1	0		
13 × 10 × 3	MRAS					12	11	10	9	8	7	6	5	4	3	2	1	0															
	MBA																		2	1	0												
	MCAS																					9	8	7	6	5	4	3	2	1	0		
13 × 10 × 2	MRAS					12	11	10	9	8	7	6	5	4	3	2	1	0															
	MBA																		1	0													
	MCAS																					9	8	7	6	5	4	3	2	1	0		
13 × 9 × 2	MRAS					12	11	10	9	8	7	6	5	4	3	2	1	0															
	MBA																		1	0													
	MCAS																					8	7	6	5	4	3	2	1	0			

Chip select interleaving is supported for the memory controller, and is programmed in DDR_SDRAM_CFG[BA_INTLV_CTL]. Interleaving is supported between chip selects 0 and 1. When interleaving is enabled, the chip selects being interleaved must use the same size of memory. One extra bit in the address decode is used for the interleaving to determine which chip select to access.

Table 10-40 illustrates examples of address decode when interleaving between two chip selects.

Table 10-40. Example of Address Multiplexing for -Bit Data Bus Interleaving between Two Banks with Partial Array Self Refresh Disabled

14 × 10 × 3	MRAS			13	12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL																					
	MBA																		CS SEL	2	1	0																	
	MCAS																						9	8	7	6	5	4	3	2	1	0							
14 × 10 × 2	MRAS			13	12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL																					
	MBA																		CS SEL	1	0																		
	MCAS																						9	8	7	6	5	4	3	2	1	0							
13 × 10 × 3	MRAS			12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL																						
	MBA																		CS SEL	2	1	0																	
	MCAS																						9	8	7	6	5	4	3	2	1	0							
13 × 10 × 2	MRAS			12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL																						
	MBA																		CS SEL	1	0																		
	MCAS																						9	8	7	6	5	4	3	2	1	0							

10.5.3 JEDEC Standard DDR SDRAM Interface Commands

The following section describes the commands and timings the controller uses when operating in DDR2 mode.

All read or write accesses to DDR SDRAM are performed by the DDR memory controller using JEDEC standard DDR SDRAM interface commands. The SDRAM device samples command and address inputs on rising edges of the memory clock; data is sampled using both the rising and falling edges of DQS. Data read from the DDR SDRAM is also sampled on both edges of DQS.

The following DDR SDRAM interface commands (summarized in Table 10-41) are provided by the DDR controller. All actions for these commands are described from the perspective of the SDRAM device.

- Row activate
Latches row address and initiates memory read of that row. Row data is latched in SDRAM sense amplifiers and must be restored by a precharge command before another row activate occurs.
- Precharge
Restores data from the sense amplifiers to the appropriate row. Also initializes the sense amplifiers in preparation for reading another row in the memory array (performing another activate command). Precharge must occur after read or write, if the row address changes on the next open page mode access.
- Read
Latches column address and transfers data from the selected sense amplifier to the output buffer as determined by the column address. During each succeeding clock edge, additional data is driven without additional read commands. The amount of data transferred is determined by the burst size which defaults to 4.

- **Write**
Latches column address and transfers data from the data pins to the selected sense amplifier as determined by the column address. During each succeeding clock edge, additional data is transferred to the sense amplifiers from the data pins without additional write commands. The amount of data transferred is determined by the data masks and the burst size, which is set to four by the DDR memory controller.
- **Refresh (similar to \overline{MCAS} before \overline{MRAS})**
Causes a row to be read in all logical banks (JEDEC SDRAM) as determined by the refresh row address counter. This refresh row address counter is internal to the SDRAM. After being read, the row is automatically rewritten in the memory array. All logical banks must be in a precharged state before executing a refresh. The memory controller also supports posted refreshes, where several refreshes may be executed at once, and the refresh interval may be extended.
- **Mode register set (for configuration)**
Allows setting of DDR SDRAM options. These options are: \overline{MCAS} latency, additive latency, write recovery, burst type, and burst length. \overline{MCAS} latency may be chosen as provided by the preferred SDRAM (some SDRAMs provide \overline{MCAS} latency {1,2,3}, some provide \overline{MCAS} latency {1,2,3,4,5}, and so on). Burst type is always sequential. Although some SDRAMs provide burst lengths of 1, 2, 4, 8, and page size, this memory controller supports a burst length of 4. For DDR2 in 32-bit bus mode, all 32-byte burst accesses from the platform are split into two 16-byte (that is, 4-beat) accesses to the SDRAMs in the memory controller. The mode register set command is performed by the DDR memory controller during system initialization. Parameters such as mode register data, \overline{MCAS} latency, burst length, and burst type, are set by software in `DDR_SDRAM_MODE[SDMODE]` and transferred to the SDRAM array by the DDR memory controller after `DDR_SDRAM_CFG[MEM_EN]` is set. If `DDR_SDRAM_CFG[BI]` is set to bypass the automatic initialization, then the MODE registers can be configured through software through use of the `DDR_SDRAM_MD_CNTL` register.
- **Self refresh (for long periods of standby)**
Used when the device is in standby for very long periods of time. Automatically generates internal refresh cycles to keep the data in all memory banks refreshed. Before execution of this command, the DDR controller places all logical banks in a precharged state.

Table 10-41. DDR SDRAM Command Table

Operation	CKE Prev.	CKE Current	\overline{MCS}	\overline{MRAS}	\overline{MCAS}	\overline{MWE}	MBA	MA10	MA
Activate	H	H	L	L	H	H	Logical bank select	Row	Row
Precharge select logical bank	H	H	L	L	H	L	Logical bank select	L	X
Precharge all logical banks	H	H	L	L	H	L	X	H	X
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

Table 10-41. DDR SDRAM Command Table (continued)

Operation	CKE Prev.	CKE Current	$\overline{\text{MCS}}$	$\overline{\text{MRAS}}$	$\overline{\text{MCAS}}$	$\overline{\text{MWE}}$	MBA	MA10	MA
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

10.5.4 DDR SDRAM Interface Timing

The DDR memory controller supports four-beat bursts to SDRAM. For single-beat reads, the DDR memory controller performs a four-beat burst read, but ignores the last three beats. Single-beat writes are performed by masking the last three beats of the four-beat burst using the data mask MDM[0:3]. If ECC is disabled, writes smaller than double words are performed by appropriately activating the data mask. If ECC is enabled, the controller performs a read-modify write.

NOTE

If a second read or write is pending, reads shorter than four beats are not terminated early even if some data is irrelevant.

To accommodate available memory technologies across a wide spectrum of operating frequencies, the DDR memory controller allows the setting of the intervals defined in [Table 10-42](#) with granularity of one memory clock cycle, except for CASLAT, which can be programmed with 1/2 clock granularity.

Table 10-42. DDR SDRAM Interface Timing Intervals

Timing Intervals	Definition
ACTTOACT	The number of clock cycles from a bank-activate command until another bank-activate command within a physical bank. This interval is listed in the AC specifications of the SDRAM as t_{RRD} .
ACTTOPRE	The number of clock cycles from an activate command until a precharge command is allowed. This interval is listed in the AC specifications of the SDRAM as t_{RAS} .
ACTTORW	The number of clock cycles from an activate command until a read or write command is allowed. This interval is listed in the AC specifications of the SDRAM as t_{RCD} .
BSTOPRE	The number of clock cycles to maintain a page open after an access. The page open duration counter is reloaded with BSTOPRE each time the page is accessed (including page hits). When the counter expires, the open page is closed with an SDRAM precharge bank command as soon as possible.
CASLAT	Used in conjunction with additive latency to obtain the READ latency. The number of clock cycles between the registration of a READ command by the SDRAM and the availability of the first piece of output data. If a READ command is registered at clock edge n , and the read latency is m clocks, the data is available nominally coincident with clock edge $n + m$.
PRETOACT	The number of clock cycles from a precharge command until an activate or a refresh command is allowed. This interval is listed in the AC specifications of the SDRAM as t_{RP} .

Table 10-42. DDR SDRAM Interface Timing Intervals (continued)

Timing Intervals	Definition
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 10.4.1.4, “DDR SDRAM Timing Configuration 0 (TIMING_CFG_0),” Section 10.4.1.5, “DDR SDRAM Timing Configuration 1 (TIMING_CFG_1),” Section 10.4.1.6, “DDR SDRAM Timing Configuration 2 (TIMING_CFG_2),” and Section 10.4.1.3, “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 10-35 through Figure 10-37 show DDR SDRAM timing for various types of accesses; see Figure 10-35 for a single-beat read operation, Figure 10-36 for a single-beat write operation, and Figure 10-37 for a double word write operation. Note that all signal transitions occur on the rising edge of the memory bus clock and that single-beat read operations are identical to burst-reads. These figures

assume the CLK_ADJUST is set to 1/2 DRAM cycle, an additive latency of 0 DRAM cycles is used, and the write latency is 1 DRAM cycle.

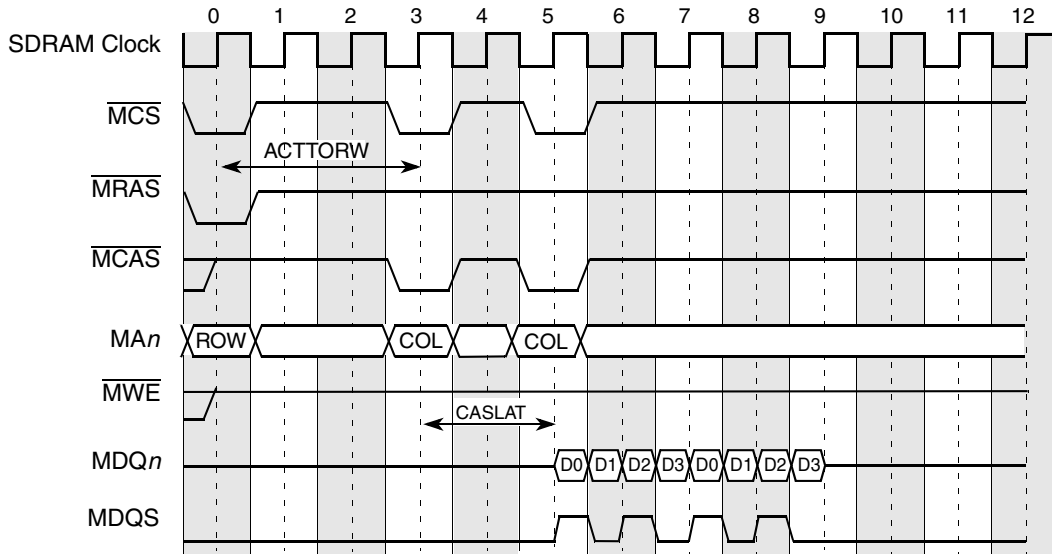


Figure 10-35. DDR SDRAM Burst Read Timing—ACTTORW = 3, MCAS Latency = 2

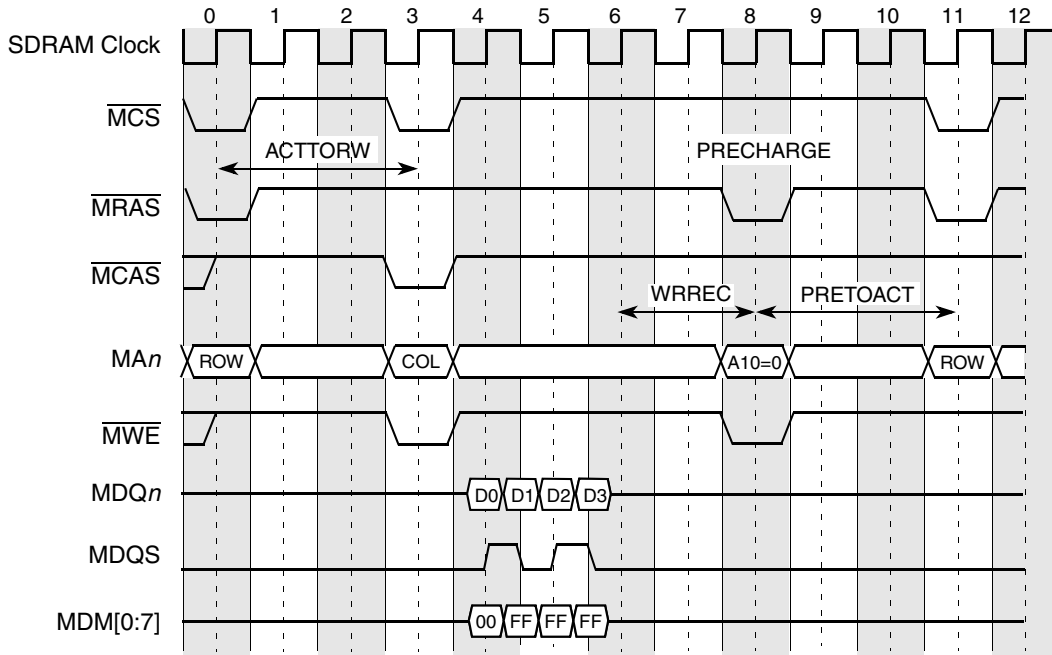


Figure 10-36. DDR SDRAM Single-Beat (Double Word) Write Timing—ACTTOR

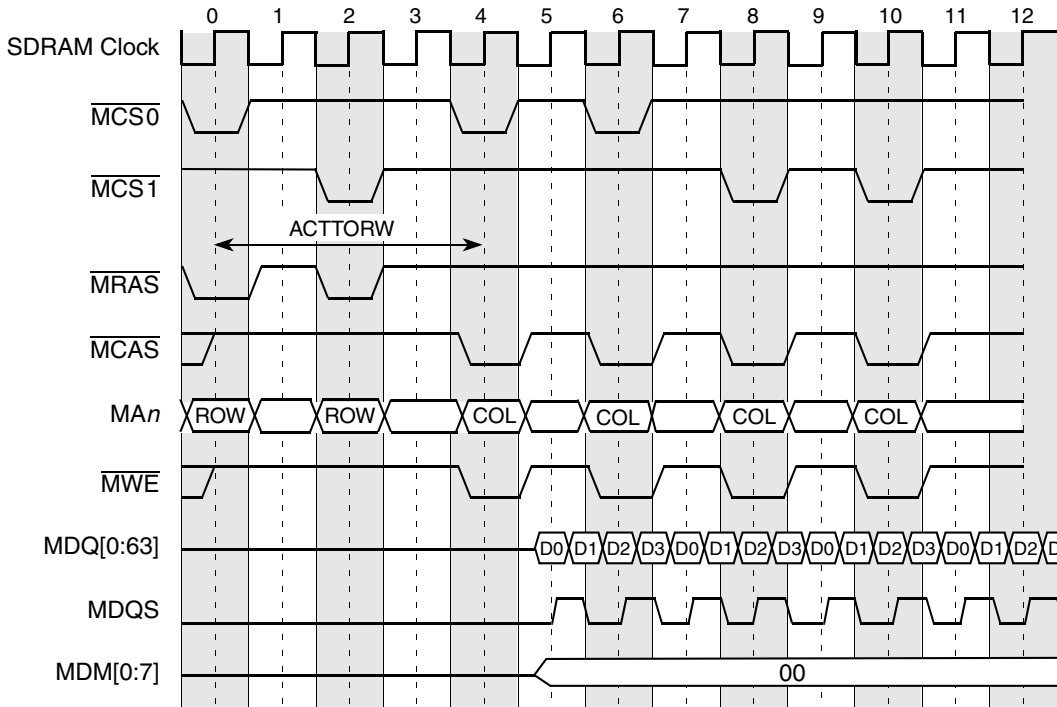


Figure 10-37. DDR SDRAM Single-Beat (Double Word) Write Timing—ACTTORW = 3

10.5.4.1 Clock Distribution

The following list discusses recommendations for clock distribution.

- If running with many devices, zero-delay PLL clock buffers, JEDEC-JESD82 standard, should be used. These buffers were designed for DDR applications.
- PCB traces for DDR clock signals should be short, all on the same layer, and of equal length and loading.

DDR SDRAM manufacturers provide detailed information on PCB layout and termination issues.

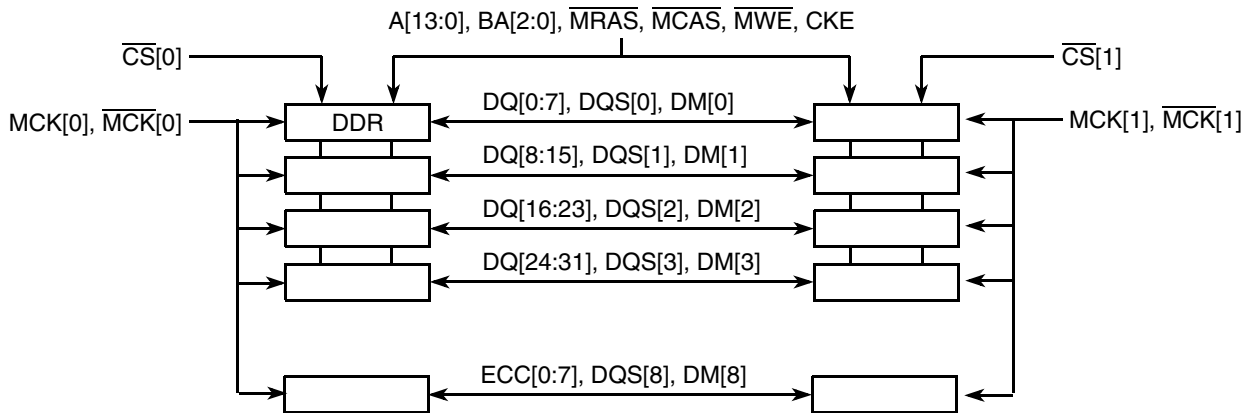


Figure 10-38. DDR SDRAM Clock Distribution Example for ×8 DDR SDRAMs

10.5.5 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 10-39 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.

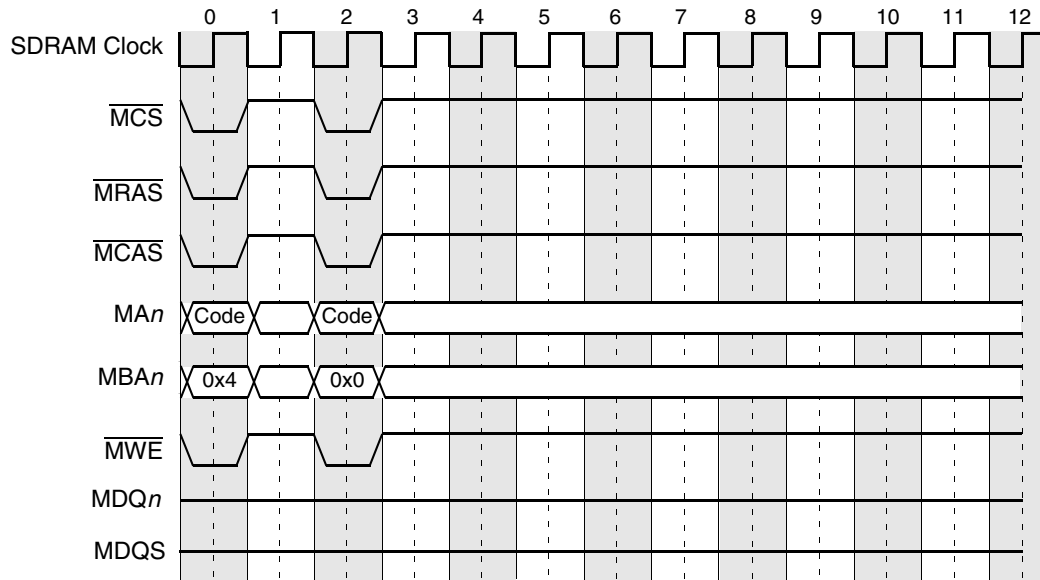


Figure 10-39. DDR SDRAM Mode-Set Command Timing

10.5.6 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 10-40 shows the registered DDR SDRAM DIMM single-beat write timing.

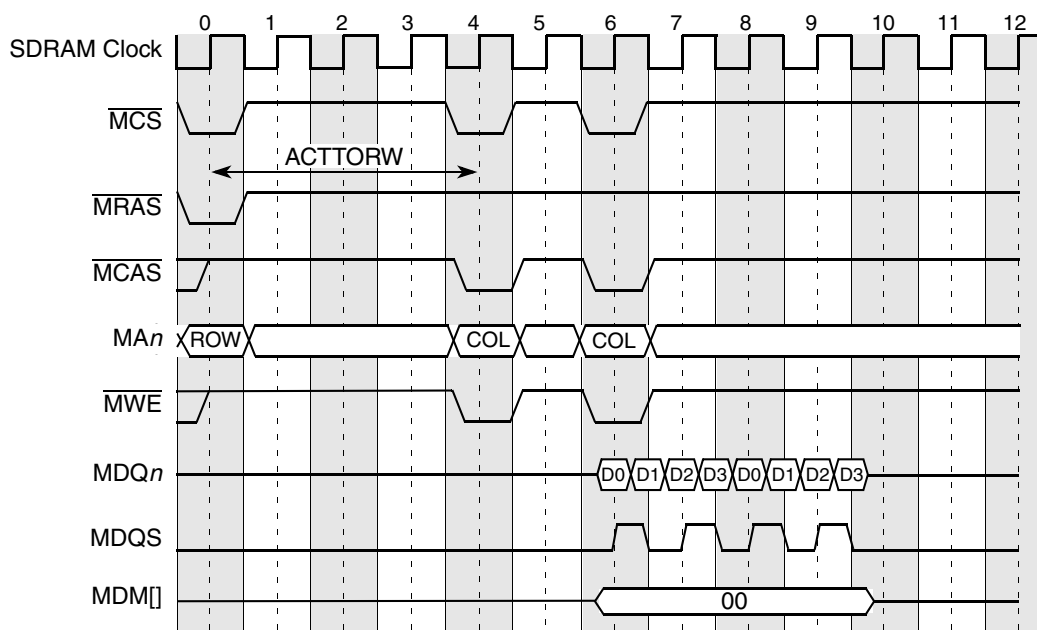


Figure 10-40. Registered DDR SDRAM DIMM Burst Write Timing

10.5.7 DDR SDRAM Write Timing Adjustments

The DDR memory controller facilitates system design flexibility by providing a write timing adjustment parameter, write data delay, (TIMING_CFG_2[WR_DATA_DELAY]) for data and DQS. The DDR SDRAM specification requires DQS be received no sooner than 75% of an SDRAM clock period—and no later than 125% of a clock period—from the capturing clock edge of the command/address at the SDRAM. TIMING_CFG_2[WR_DATA_DELAY] specifies how much to delay the launching of DQS and data from the first clock edge occurring one SDRAM clock cycle after the command is launched. The delay increment step sizes are in 1/4 SDRAM clock periods starting with the default value of 0.

Figure 10-41 shows the use of the WR_DATA_DELAY parameter.

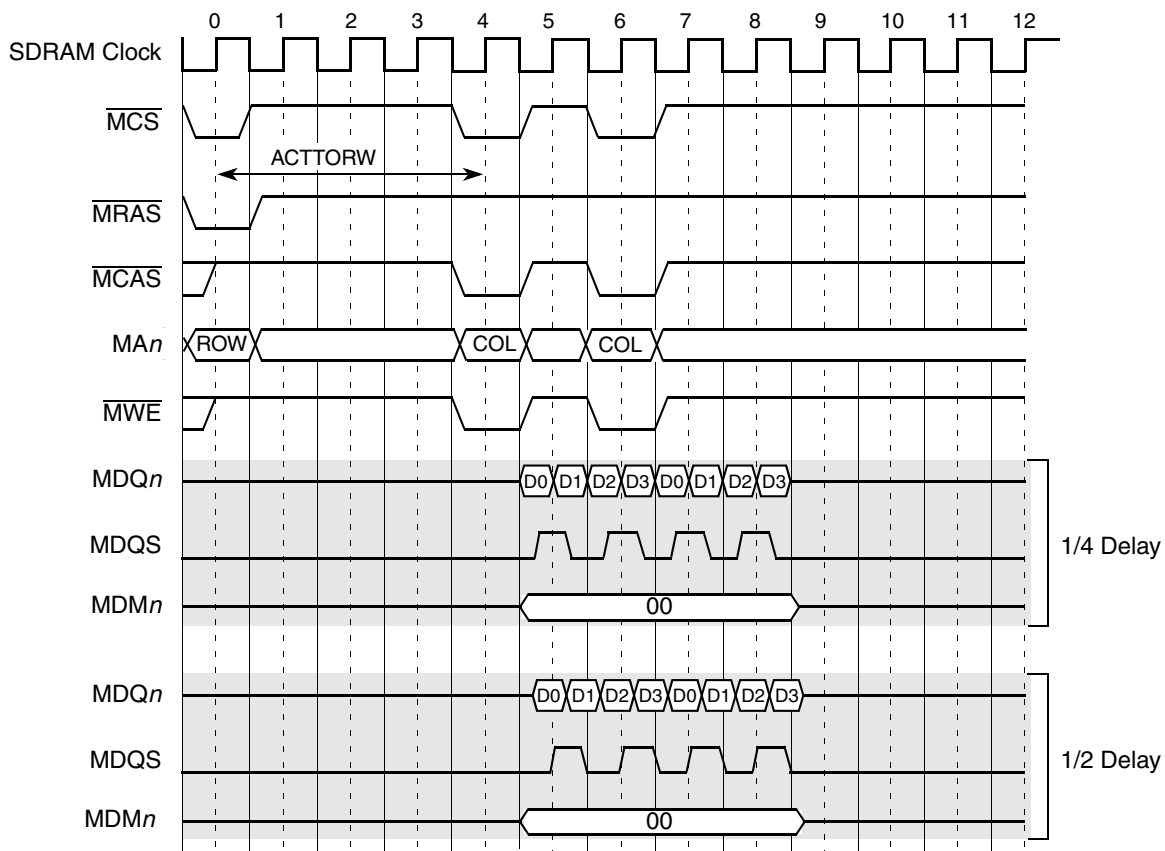


Figure 10-41. Write Timing Adjustments Example for Write Latency = 1

10.5.8 DDR SDRAM Refresh

The DDR memory controller supports auto-refresh and self-refresh. Auto refresh is used during normal operation and is controlled by the `DDR_SDRAM_INTERVAL[REFINT]` value; self-refresh is used only when the DDR memory controller is set to enter a sleep power management state. The REFINT value, which represents the number of memory bus clock cycles between refresh cycles, must allow for possible outstanding transactions to complete before a refresh request is sent to the memory after the REFINT value is reached. If a memory transaction is in progress when the refresh interval is reached, the refresh cycle waits for the transaction to complete. In the worst case, the refresh cycle must wait the number of bus clock cycles required by the longest programmed access. To ensure that the latency caused by a memory transaction does not violate the device refresh period, it is recommended that the programmed value of REFINT be less than that required by the SDRAM.

When a refresh cycle is required, the DDR memory controller does the following:

1. Completes all current memory requests.
2. Closes all open pages with a PRECHARGE-ALL command to each DDR SDRAM bank with an open page (as indicated by the row open table).
3. Issues one or more auto-refresh commands to each DDR SDRAM bank (as identified by its chip select) to refresh one row in each logical bank of the selected physical bank.

The auto-refresh commands are staggered across the two possible banks to reduce the system’s instantaneous power requirements. Three sets of auto refresh commands are issued on consecutive cycles when the memory is populated with one 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 two banks are installed. After the refresh sequence completes, any pending memory request is initiated after an inactive period specified by TIMING_CFG_1 [REFREC] and TIMING_CFG_3[EXT_REFREC]. In addition, posted refreshes are supported to allow the refresh interval to be set to a larger value.

10.5.8.1 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 is allowed. The DDR memory controller implements bank staggering for refreshes, as shown in Figure 10-42 (TIMING_CFG_1 [REFREC] = 10 in this example).

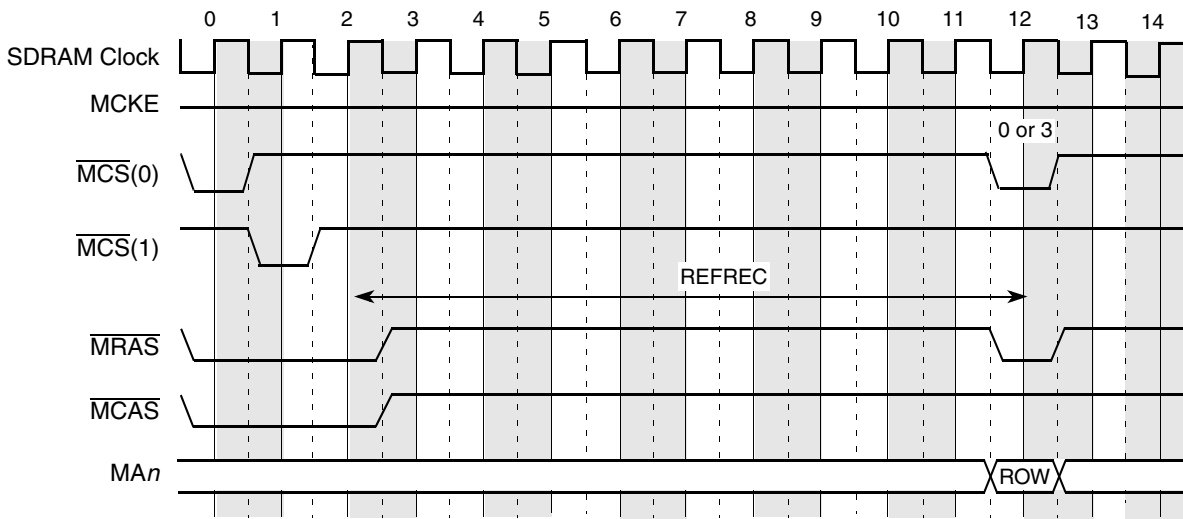


Figure 10-42. DDR SDRAM Bank Staggered Auto Refresh Timing

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.

10.5.8.2 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 10-43 summarizes the refresh types available in each power-saving mode.

Table 10-43. DDR SDRAM Power-Saving Modes Refresh Configuration

Power Saving Mode	Refresh Type	SREN
Sleep	Self	1
	None	—

Note that in the absence of refresh support, system software must preserve DDR SDRAM data (such as by copying the data to disk) before entering the power-saving mode.

The dynamic power-saving mode uses the CKE DDR SDRAM pin to dynamically power down when there is no system memory activity. The CKE pin is negated when both of the following conditions are met:

- No memory refreshes are scheduled
- No memory accesses are scheduled

CKE is reasserted when a new access or refresh is scheduled or the dynamic power mode is disabled. This mode is controlled with DDR_SDRAM_CFG[DYN_PWR_MGMT].

Dynamic power management mode offers tight control of the memory system's power consumption by trading power for performance through the use of CKE. Powering up the DDR SDRAM when a new memory reference is scheduled causes an access latency penalty, depending on whether active or precharge powerdown is used, along with the settings of TIMING_CFG_0[ACT_PD_EXIT] and TIMING_CFG_0[PRE_PD_EXIT]. A penalty of 1 cycle is shown in Figure 10-43.

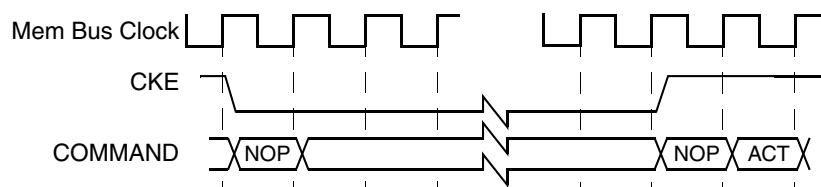


Figure 10-43. DDR SDRAM Power-Down Mode

10.5.8.2.1 Self-Refresh in Sleep Mode

The entry and exit timing for self-refreshing SDRAMs is shown in Figure 10-44 and Figure 10-45.

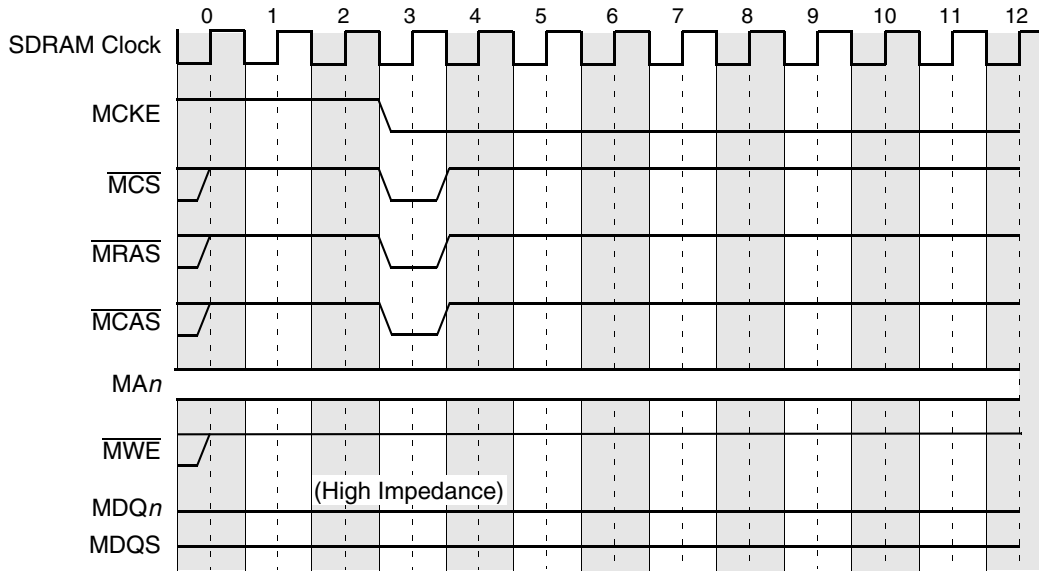


Figure 10-44. DDR SDRAM Self-Refresh Entry Timing

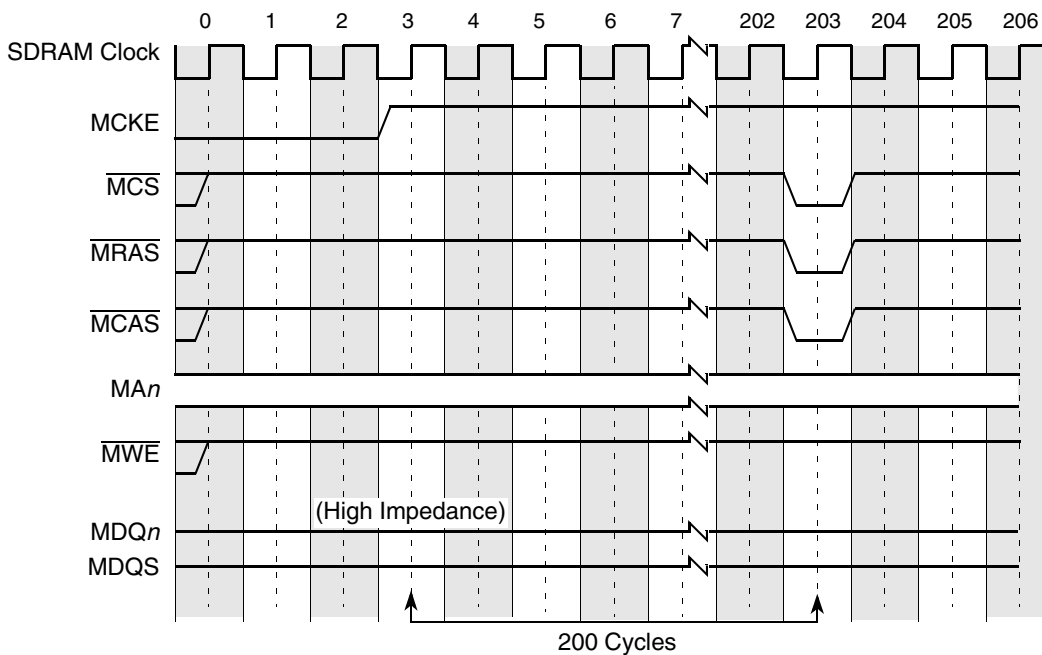


Figure 10-45. DDR SDRAM Self-Refresh Exit Timing

10.5.9 DDR Data Beat Ordering

Transfers to and from memory are always performed in four-beat bursts (four beats = 16 bytes when a 32-bit bus is used). For transfer sizes other than four beats, the data transfers are still operated as four-beat bursts. If ECC is enabled and either the access is not doubleword aligned or the size is not a multiple of a

doubleword, a full read-modify-write is performed for a write to SDRAM. If ECC is disabled or both the access is doubleword aligned with a size that is a multiple of a doubleword, the data masks (MDM[0:4] for 32-bit bus) can be used to prevent the writing of unwanted data to SDRAM. The DDR memory controller also uses data masks to prevent all unintended full double words from writing to SDRAM. For example, if a write transaction is desired with a size of one word (4 bytes), then the second, third, and fourth beats of data are not written to DRAM (assuming a 32-bit data bus).

Table 10-44 lists the data beat sequencing to and from the DDR SDRAM and the data queues for each of the possible transfer sizes with each of the possible starting double-word offsets. All underlined double-word offsets are valid for the transaction.

Table 10-44. Memory Controller–Data Beat Ordering

Transfer Size	Starting Double-Word Offset	Double-Word Sequence ¹ to/from DRAM and Queues
1 double word	0	<u>0</u> - 1 - 2 - 3
	1	1 - 2 - 3 - 0
	2	<u>2</u> - 3 - 0 - 1
	3	<u>3</u> - 0 - 1 - 2
2 double words	0	<u>0</u> - <u>1</u> - 2 - 3
	1	<u>1</u> - <u>2</u> - 3 - 0
	2	<u>2</u> - <u>3</u> - 0 - 1
3 double words	0	<u>0</u> - <u>1</u> - <u>2</u> - 3
	1	<u>1</u> - <u>2</u> - <u>3</u> - 0

¹ All underlined and bolded Double-word offsets are valid for the transaction.

10.5.10 Page Mode and Logical Bank Retention

The DDR memory controller supports an open/closed page mode with an allowable open page for each logical bank of DRAM used. In closed page mode for DDR SDRAMs, the DDR memory controller uses the SDRAM auto-precharge feature, which allows the controller to indicate that the page must be automatically closed by the DDR SDRAM after the READ or WRITE access. This is performed using MA[10] of the address during the COMMAND phase of the access to enable auto-precharge. Auto-precharge is non-persistent in that it is either enabled or disabled for each individual READ or WRITE command. It can, however, be enabled or disabled separately for each chip select.

When the DDR memory controller operates in open page mode, it retains the currently active SDRAM page by not issuing a precharge command. The page remains opens until one of the following conditions occurs:

- Refresh interval is met.
- The user-programmable DDR_SDRAM_INTERVAL[BSTOPRE] value is exceeded.
- There is a logical bank row collision with another transaction that must be issued.

Page mode can dramatically reduce access latencies for page hits. Depending on the memory system design and timing parameters, using page mode can save two to three clock cycles for subsequent burst accesses that hit in an active page. Also, better performance can be obtained using more banks, especially

in systems which use many different channels. Page mode is disabled by clearing DDR_SDRAM_INTERVAL[BSTOPRE] or setting CS_n_CONFIG[AP_nEN].

10.5.11 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 10-45](#) and [Table 10-46](#).

In 32-bit mode, [Table 10-45](#) is split into 2 halves. The first half, consisting of rows 0–31, is used to calculate the ECC bits for the first 32 data bits of any 64-bit granule of data. This always applies to the odd data beats on the DDR data bus. The second half of the table, consisting of rows 32–63, is used to calculate the ECC bits for the second 32 bits of any 64-bit granule of data. This always applies to the even data beats on the DDR data bus.

Table 10-45. DDR SDRAM ECC Syndrome Encoding

Data Bit	Syndrome Bit							
	0	1	2	3	4	5	6	7
0	•	•						•
1	•		•					•
2	•			•				•
3	•				•			•
4	•	•				•		
5	•		•			•		
6	•			•		•		
7	•				•	•		
8	•	•					•	
9	•		•				•	
10	•			•			•	
11	•				•		•	
32			•	•				•
33			•		•			•
34	•		•		•			
35		•	•		•			
36			•	•		•		
37			•		•	•		
38	•		•		•	•		•
39		•	•		•	•		•
40			•	•			•	
41			•		•		•	
42	•		•		•		•	•
43		•	•		•		•	•

Table 10-45. DDR SDRAM ECC Syndrome Encoding (continued)

Data Bit	Syndrome Bit								Data Bit	Syndrome Bit							
	0	1	2	3	4	5	6	7		0	1	2	3	4	5	6	7
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 10-46. DDR SDRAM ECC Syndrome Encoding (Check Bits)

Check Bit	Syndrome Bit							
	0	1	2	3	4	5	6	7
0	•							
1		•						
2			•					
3				•				
4					•			
5						•		
6							•	
7								•

10.5.12 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 10.4.1.26, “Memory Error Interrupt Enable \(ERR_INT_EN\),”](#) [Section 10.4.1.25, “Memory Error Disable \(ERR_DISABLE\),”](#) and [Section 10.4.1.24, “Memory Error Detect \(ERR_DETECT\).”](#)

Multi-bit errors can generate an MCP while single-bit errors generate a normal interrupt to the e300 core. Single-bit errors are counted and reported based on the ERR_SBE value. When a single-bit error is detected, the DDR memory controller does the following:

- Corrects the data
- Increments the single-bit error counter ERR_SBE[SBEC]
- Generates a critical interrupt if the counter value ERR_SBE[SBEC] equals the programmable threshold ERR_SBE[SBET]
- Completes the transaction normally

If a multi-bit error is detected for a read, the DDR memory controller logs the error and generates the interrupt (if enabled, as described in [Section 10.4.1.25, “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 10.4.1.24, “Memory Error Detect \(ERR_DETECT\)”](#)). This error is detected if the address from the memory request does not fall into any of the enabled, programmed chip select address ranges.

[Table 10-47](#) shows the errors with their descriptions. The final error the memory controller detects is the automatic calibration error. This error is set if the memory controller detects an error during its training sequence.

Table 10-47. Memory Controller Errors

Error	Descriptions	Action	Detect Register
Single-bit ECC threshold	The number of ECC errors has reached the threshold specified in the ERR_SBE.	The error is reported through interrupt if enabled.	The error control register only logs read versus write
Multi-bit ECC error	A multi-bit ECC error is detected during a read, or read-modify-write memory operation.		
Memory select error	Read, or write, address does not fall within the address range of any of the memory banks.		

10.6 Initialization/Application Information

System software must configure the DDR memory controller, using a memory polling algorithm at system start-up, to correctly map the size of each bank in memory. Then, the DDR memory controller uses its bank map to assert the appropriate \overline{MCS}_n signal for memory accesses according to the provided bank depths. System software must also configure the DDR memory controller at system start-up to appropriately multiplex the row and column address bits for each bank. Refer to row-address configuration in [Section 10.4.1.2, “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 10.4.1, “Register Descriptions,”](#) for more detailed descriptions of the configuration registers. These parameters are shown in [Table 10-48](#).

Table 10-48. Memory Interface Configuration Register Initialization Parameters

Name	Description	Parameter	Section/page
CS _n _BNDS	Chip select memory bounds	SA _n EA _n	10.4.1.1/10-1
CS _n _CONFIG	Chip select configuration	CS _n _EN AP _n _EN ODT_RD_CFG ODT_WR_CFG	10.4.1.2/10-1 10.4.1.2/10-1 10.4.1.2/10-1
TIMING_CFG_3	Extended timing parameters for fields in TIMING_CFG_1	EXT_REFREC	10.4.1.3/10-1 10.4.1.3/10-1
TIMING_CFG_0	Timing configuration	RWT WRT RRT WWT	10.4.1.4/10-1 10.4.1.4/10-1 10.4.1.4/10-1 10.4.1.4/10-1
TIMING_CFG_1	Timing configuration	PRETOACT ACTTOPRE ACTTORW CASLAT	10.4.1.5/10-1 10.4.1.5/10-1 10.4.1.5/10-1 10.4.1.5/10-1
TIMING_CFG_2	Timing configuration	ADD_LAT CPO WR_LAT RD_TO_PRE	10.4.1.6/10-1 10.4.1.6/10-1 10.4.1.6/10-1 10.4.1.6/10-1
DDR_SDRAM_CFG	Control configuration	SREN RD_EN SDRAM_TYPE DYN_PWR DBW	10.4.1.7/10-1 10.4.1.7/10-1 10.4.1.7/10-1 10.4.1.7/10-1 10.4.1.7/10-1
DDR_SDRAM_CFG_2	Control configuration	DLL_RST_DIS DQS_CFG ODT_CFG	10.4.1.8/10-2 10.4.1.8/10-2 10.4.1.8/10-2
DDR_SDRAM_MODE	Mode configuration	ESDMODE SDMODE	10.4.1.9/10-2 10.4.1.9/10-2
DDR_SDRAM_MODE_2	Mode configuration	ESDMODE2 ESDMODE3	10.4.1.10/10-24 10.4.1.10/10-24
DDR_SDRAM_INTERVAL	Interval configuration	REFINT BSTOPRE	10.4.1.12/10-27 10.4.1.12/10-27
DDR_DATA_INIT	Data initialization configuration register	INIT_VALUE	10.4.1.13/10-28 10.4.1.13/10-28
DDR_SDRAM_CLK_CNTL	Clock adjust	CLK_ADJUST	10.4.1.14/10-28 10.4.1.14/10-28
DDR_INIT_ADDR	Initialization address	INIT_ADDR	10.4.1.15/10-29 10.4.1.15/10-29

10.6.1 DDR SDRAM Initialization Sequence

After configuration of all parameters is complete, system software must set `DDR_SDRAM_CFG[MEM_EN]` to enable the memory interface. Note that 200 μ s must elapse after DRAM clocks are stable (`DDR_SDRAM_CLK_CNTL[CLK_ADJUST]` is set and any chip select is enabled) before `MEM_EN` can be set, so a delay loop in the initialization code may be necessary if software is enabling the memory controller. If `DDR_SDRAM_CFG[BI]` is not set, the DDR memory controller conducts an automatic initialization sequence to the memory, which follows the memory specifications. If the bypass initialization mode is used, then software can initialize the memory through the `DDR_SDRAM_MD_CNTL` register.

Chapter 11

Enhanced Local Bus Controller

This chapter describes the enhanced local bus controller (eLBC) block. It describes the external signals and the memory-mapped registers as well as a functional description of the general-purpose chip-select machine (GPCM), NAND Flash control machine (FCM), and user-programmable machines (UPMs) of the eLBC. Finally, it includes an initialization and applications information section with many specific examples of its use.

11.1 Introduction

Figure 11-1 is a functional block diagram of the eLBC, which supports three interfaces: GPCM, FCM, and UPM controllers.

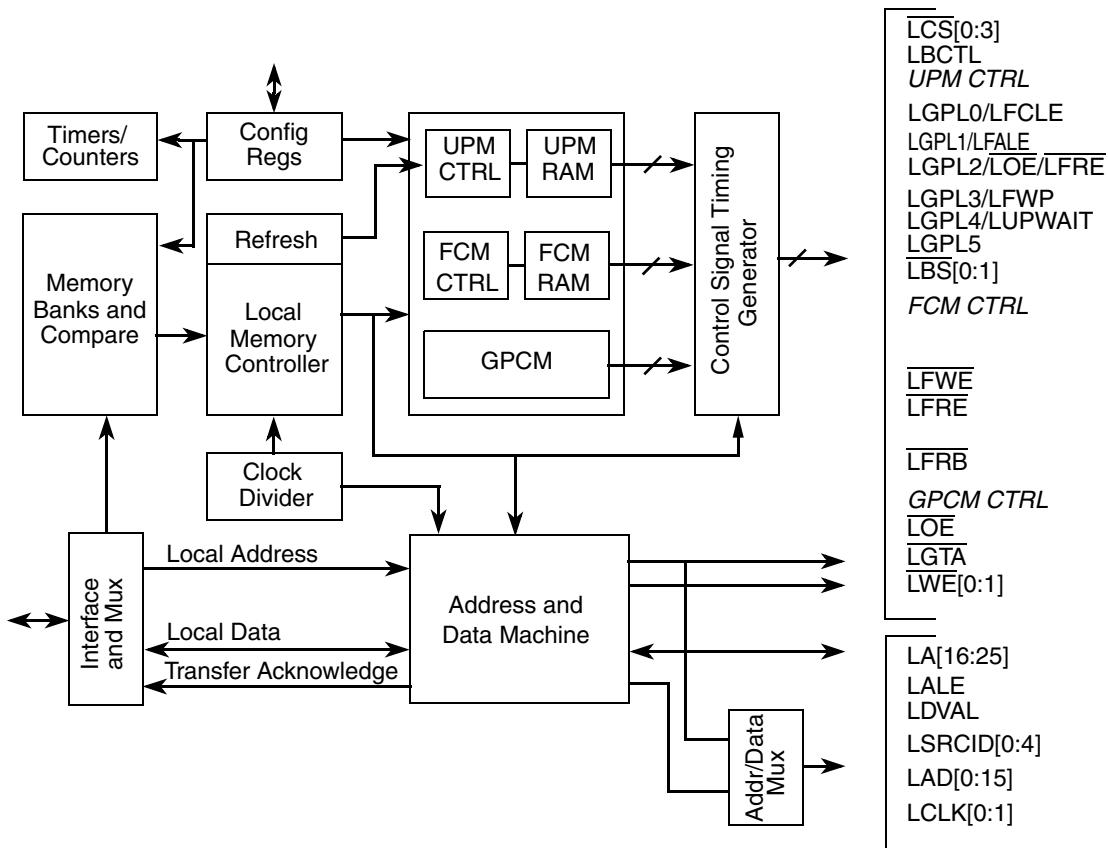


Figure 11-1. Enhanced Local Bus Controller Block Diagram

11.1.1 Overview

The main component of the eLBC is its memory controller, which provides a seamless interface to many types of memory devices and peripherals. The memory controller is responsible for controlling four memory banks shared by a GPCM, an FCM, and up to three UPMs. As such, it supports a minimal glue logic interface to SRAM, EPROM, NOR Flash EEPROM, NAND Flash EEPROM, burstable RAM, regular DRAM devices, extended data output DRAM devices, and other peripherals. The external address latch signal (LALE) allows multiplexing of addresses with data signals to reduce the device pin count. The eLBC also includes a number of data checking and protection features such as write protection, and a bus monitor to ensure that each bus cycle is terminated within a user-specified period.

11.1.2 Features

The eLBC main features are as follows:

- Memory controller with four memory banks
 - 32-bit address decoding with mask
 - Variable memory block sizes (64Kbytes to 2 Gbytes in FCM mode, 32 Kbytes to 64 Mbytes in GPCM and UPM modes)
 - Selection of control signal generation on a per-bank basis
 - Data buffer controls activated on a per-bank basis
 - Automatic segmentation of large transactions into memory accesses optimized for bus width and addressing capability
 - Write-protection capability
- General-purpose chip-select machine (GPCM)
 - Compatible with SRAM, EPROM, NOR Flash EEPROM, and peripherals
 - Global (boot) chip-select available at system reset
 - Boot chip-select support for 8- and 16-bit devices
 - Minimum three-clock access to external devices
 - Two byte-write-enable signals ($\overline{\text{LWE}}[0:1]$)
 - Output enable signal ($\overline{\text{LOE}}$)
 - External access termination signal ($\overline{\text{LGTA}}$)
- NAND Flash control machine (FCM)
 - Compatible with small (512 + 16 bytes) and large (2048 + 64 bytes) page parallel NAND Flash EEPROM
 - Global (boot) chip-select available at system reset, with 4-Kbyte boot block buffer for execute-in-place boot loading
 - Boot chip-select support for 8-bit devices
 - Dual 2-Kbyte/eight 512-byte buffers allow simultaneous data transfer during flash reads and programming
 - Interrupt-driven block transfer for reads and writes
 - Programmable command and data transfer sequences of up to eight steps supported

- Generic command and address registers support proprietary flash interfaces
- Block write locking to ensure system security and integrity
- Three user-programmable machines (UPMs)
 - Programmable-array-based machine controls external signal timing with a granularity of up to one quarter of an external bus clock period
 - User-specified control-signal patterns run when an internal master requests a single-beat or burst read or write access.
 - UPM refresh timer runs a user-specified control signal pattern to support refresh
 - User-specified control-signal patterns can be initiated by software
 - Each UPM can be defined to support DRAM devices with depths of 64, 128, 256, and 512 Kbytes, and 1, 2, 4, 8, 16, 32, 64, 128, and 256 Mbytes
 - Support for 8- and 16-bit devices
 - Page mode support for successive transfers within a burst
 - Internal address multiplexing supporting 64-, 128-, 256-, and 512-Kbyte, and 1-, 2-, 4-, 8-, 16-, 32-, 64-, 128-, and 256-Mbyte page banks
- Optional monitoring of transfers between local bus internal masters and local bus slaves (local bus error reporting on interrupt and status registers)
- Different machines (FCM/GPCM/UPM) share the address, data, and control signals. While the eLBC is servicing a transaction, subsequent transactions are queued until the current transaction has completed.

11.1.3 Modes of Operation

The eLBC provides one GPCM, one FCM, and three UPMs for the local bus, with no restriction on how many of the four banks (chip selects) can be programmed to operate with any given machine. The internal transaction address is limited to 32 bits, so all chip selects must fall within the 4-Gbyte window addressed by the internal transaction address. When a memory transaction is dispatched to the eLBC, the internal transaction address is compared with the address information of each bank (chip select). The corresponding machine assigned to that bank (GPCM, FCM, or UPM) then takes ownership of the external signals that control the access and maintains control until the transaction ends. Thus, with the eLBC in GPCM or FCM, or UPM mode, only one of the four chip selects is active at any time for the duration of the transaction except in the case of UPM refresh where all UPM machines that are enabled for refresh have concurrent chip select assertion.

11.1.3.1 eLBC Bus Clock and Clock Ratios

The eLBC supports ratios of 2, 4, and 8 between the faster internal (CSB) clock and slower external bus clock (LCLK[0:1]). This ratio is software programmable through the clock ratio register (LCRR[CLKDIV]). This ratio affects the resolution of signal timing shifts in GPCM and FCM modes and the interpretation of UPM array words in UPM mode. The bus clock is driven identically onto pins, LCLK[0:1], to allow the clock load to be shared equally across a set of signal nets, thereby enhancing the edge rates of the bus clock.

11.1.3.2 Source ID Debug Mode

The eLBC provides the ID of a transaction source on external device pins. When those pins are selected, the 5-bit internal ID of the current transaction source appears on LSRCID[0:4] whenever valid address or data is available on the eLBC external pins. The reserved value of 0x1F, which indicates invalid address or data, appears on the source ID pins at all other times. The combination of a valid source ID (any value except 0x1F) and the value of external address latch enable (LALE) and data valid (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 26-bit address may be latched from LAD[0:15] and combined with LA[16:25].
- If a valid source ID is detected on LSRCID[0:4] and LDVAL is asserted, valid data may be latched from LAD.

The LSRCID[0:4] and LDVAL signals are multiplexed with other functions sharing the same external pins. Refer to [Chapter 3, “Signal Descriptions,”](#) and [Chapter 4, “Reset, Clocking, and Initialization,”](#) to learn how to enable the LSRCID/LDVAL pins.

11.2 External Signal Descriptions

[Table 11-1](#) contains a list of external signals related to the eLBC and summarizes their function. The table also shows the reset state of all external signals during assertion of $\overline{\text{HRESET}}$. For more information on the use of some of these signals as reset configuration signals on the device, see “Power on Reset Flow.”

Table 11-1. Signal Properties—Summary

Name	Alternate Function(s)	Mode	Descriptions	No. of Signals	I/O	Reset State (Outputs)
LALE	—	—	External address latch enable	1	O	Reset_cfg
$\overline{\text{CS}}[0:3]$	—	—	Chip selects 0–3	4	O	Reset_cfg
$\overline{\text{LWE}}0/$ $\overline{\text{LWE}}/$ $\overline{\text{LBS}}0$	$\overline{\text{LWE}}0$	GPCM	Write enable 0	1	O	Reset_cfg
	$\overline{\text{LWE}}$	FCM	Write enable	1		
	$\overline{\text{LBS}}0$	UPM	Byte (lane) select 0	1		
$\overline{\text{LWE}}1/$ $\overline{\text{LBS}}1$	$\overline{\text{LWE}}$	GPCM	Write enable 1	1	O	Reset_cfg
	$\overline{\text{LBS}}$	UPM	Byte (lane) select 1	1		
LGPL0/ LFCLE	LGPL0	UPM	General purpose line 0	1	O	Reset_cfg
	LFCLE	FCM	Flash command latch enable	1		—
LGPL1/ LFALE	LGPL1	UPM	General purpose line 1	1	O	Reset_cfg
	LFALE	FCM	Flash address latch enable	1		—
$\overline{\text{LOE}}/$ LGPL2/ $\overline{\text{LFRE}}$	$\overline{\text{LOE}}$	GPCM	Output enable	1	O	—
	$\overline{\text{LFRE}}$	FCM	Flash read enable	1		—
	LGPL2	UPM	General purpose line 2	1		—

Table 11-1. Signal Properties—Summary (continued)

Name	Alternate Function(s)	Mode	Descriptions	No. of Signals	I/O	Reset State (Outputs)
LGPL3/ LFWP	LGPL3	UPM	General purpose line 3	1	O	Reset_cfg
	LFWP	FCM	Flash write protect	1		—
LGTA/ LFRB/ LGPL4/ LUPWAIT	LGTA	GPCM	Transaction termination	1	I	High-Z
	LFRB	FCM	Flash ready/busy, open-drain shared pin	1	I	—
	LGPL4	UPM	General purpose line 4	1	O	—
	LUPWAIT	UPM	External device wait	1	I	—
LGPL5	—	UPM	General purpose line 5	1	O	Reset_cfg
LBCTL	—	—	Data buffer control	1	O	—
LA[16:25]	—	—	Non-multiplexed address bus	10	O	—
LAD[0:15]	—	—	Multiplexed address/data bus	16	I/O	—
LCLK[0:1]	—	—	Local bus clocks	2	O	—
LDVAL	—	eLBC debug	Local bus data valid	1	O	—
LSRCID[0:4]	—	eLBC debug	Local bus source ID	5	O	—

Table 11-2 shows the detailed external signal descriptions for the eLBC.

Table 11-2. Enhanced 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 pins.	
		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. Note that no other control signals are asserted during the assertion of LALE.
LCS[0:3]	O	Chip selects. Four chip selects are provided that are mutually exclusive.	
		State Meaning	Asserted/Negated—Used to enable specific memory devices or peripherals connected to the eLBC. LCS[0:3] are provided on a per-bank basis with LCS0 corresponding to the chip select for memory bank 0, which has the memory type and attributes defined by BR0 and OR0.

Table 11-2. Enhanced Local Bus Controller Detailed Signal Descriptions (continued)

Signal	I/O	Description	
$\overline{\text{LWE0}}$ / $\overline{\text{LWE1}}$ / $\overline{\text{LBS0}}$ / $\overline{\text{LBS1}}$	O	GPCM write enable 0/FCM write enable/UPM byte select 0. These signals select or validate each byte lane of the data bus. For an 8-bit port size, bit 0 is the only defined signal. The least-significant address bits of each access also determine which byte lanes are considered valid for a given data transfer.	
		State Meaning	Asserted/Negated—For GPCM operation, $\overline{\text{LWE}}[0:1]$ assert for each byte lane enabled for writing. $\overline{\text{LWE}}$ enables command, address, and data writes to NAND Flash EEPROMs controlled by FCM. $\overline{\text{LBS}}[0:1]$ are programmable byte-select signals in UPM mode. See Section 11.4.4.4, “RAM Array,” for programming details about $\overline{\text{LBS}}[0:1]$.
		Timing	Assertion/Negation—See Section 11.4.2, “General-Purpose Chip-Select Machine (GPCM),” for details regarding the timing of $\overline{\text{LWE}}[0:1]$.
LGPL0/ LFCLE	O	General purpose line 0/FCM command latch enable.	
		State Meaning	Asserted/Negated—In UPM mode, LGPL0 is one of six general purpose signals; it is driven with a value programmed into the UPM array. In FCM mode, LFCLE enables command cycles to NAND Flash EEPROMs.
LGPL1/ LFALE	O	General-purpose line 1/FCM address latch enable.	
		State Meaning	Asserted/Negated—In UPM mode, LGPL1 is one of six general purpose signals; it is driven with a value programmed into the UPM array. In FCM mode, LFALE enables address cycles to NAND Flash EEPROMs.
$\overline{\text{LOE}}$ /LGPL2/ $\overline{\text{LFRE}}$	O	GPCM output enable/General-purpose line 2/FCM read enable.	
		State Meaning	Asserted/Negated—Controls the output buffer of memory when accessing memory/devices in GPCM mode. In UPM mode, LGPL2 is one of six general purpose signals; it is driven with a value programmed into the UPM array. $\overline{\text{LFRE}}$ enables data read cycles from NAND Flash EEPROMs controlled by FCM.
LGPL3/ $\overline{\text{LFWP}}$	O	General-purpose line 3/FCM write protect.	
		State Meaning	Asserted/Negated—In UPM mode, LGPL3 is one of six general purpose signals; it is driven with a value programmed into the UPM array. In FCM mode $\overline{\text{LFWP}}$ protects NAND Flash EEPROMs from accidental erasure and programming when $\overline{\text{LFWP}}$ is asserted low—see Section 11.3.1.16, “Flash Mode Register (FMR),” for programming of FCM operations to control $\overline{\text{LFWP}}$.
$\overline{\text{LGT A}}$ /LGPL4/ $\overline{\text{LFRB}}$ / LUPWAIT	I/O	GPCM transfer acknowledge/General-purpose line 4/FCM Flash ready-busy/UPM wait.	
		State Meaning	Asserted/Negated—Input in GPCM or FCM modes used for transaction termination. It may also be configured as one of six general-purpose output signals when in UPM mode or as an input to force the UPM controller to wait for the memory/device. FCM uses $\overline{\text{LFRB}}$ to stall during long-latency read and programming operations, continuing once $\overline{\text{LFRB}}$ returns high.
LGPL5	O	General-purpose line 5	
		State Meaning	Asserted/Negated—One of six general purpose signals when in UPM mode, and drives a value programmed in the UPM array.

Table 11-2. Enhanced Local Bus Controller Detailed Signal Descriptions (continued)

Signal	I/O	Description
LBCTL	O	Data buffer control. The memory controller activates LBCTL for the local bus when a GPCM-, UPM-, or FCM-controlled bank is accessed. Buffer control is disabled by setting $OR_n[BCTL D]$.
		State Meaning Asserted/Negated—The LBCTL pin normally functions as a write/ $\overline{\text{read}}$ control for a bus transceiver connected to the LAD lines. Note that an external data buffer must not drive the LAD lines in conflict with the eLBC when LBCTL is high, because LBCTL remains high after reset and during address phases.
LA[16:25]	O	Nonmultiplexed address bus. All bits driven on LA[16:25] are defined for 8-bit port sizes. For 16-bit port sizes LA[25] is a don't care.
		State Meaning Asserted/Negated—LA is the address bus used to transmit addresses to external RAM devices. Refer to Section 11.5, “Initialization/Application Information,” for address signal multiplexing.
LAD[0:15]	I/O	Multiplexed address/data bus. 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). For a port size of 8 bits, only LAD[0:7] are connected to the external RAM.
		State Meaning Asserted/Negated—LAD is the shared 16-bit address/data bus through which external RAM devices transfer data and receive addresses.
		Timing Assertion/Negation—During assertion of LALE, LAD are driven with the RAM address for the access to follow. External logic should propagate the address on LAD while LALE is asserted, and latch the address upon negation of LALE. After LALE is negated, LAD are either driven by write data or are made high-impedance by the eLBC in order to sample read data driven by an external device. Following the last data transfer of a write access, LAD are again taken into a high-impedance state.
LCLK[0:1]	O	Local bus clocks
		State Meaning Asserted/Negated—LCLK[0:1] drive an identical bus clock signal for distributed loads.
LDVAL	O	Local bus data valid (eLBC 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. For a write, LDVAL asserts for one bus cycle during the final cycle for which the current write data on LAD is valid. During burst transfers, LDVAL asserts for each data beat.
		Timing Assertion/Negation—Valid only while the eLBC is in system debug mode. In debug mode, LDVAL asserts when the eLBC generates a data transfer acknowledge.
LSRCID[0:4]	O	Local bus source ID (eLBC debug mode only). In debug mode, all LSRCID[0:4] pins are driven high unless LSRCID[0:4] is driving a debug source ID for identifying the internal system device controlling the eLBC.
		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 consists of all physical address bits—with optional padding—for reconstructing the system address presented to the eLBC.

11.3 Memory Map/Register Definition

Table 11-3 shows the memory mapped registers of the eLBC. Undefined 4-byte address spaces within offset 0x000–0xFFFF are reserved.

Table 11-3. Enhanced Local Bus Controller Registers

Enhanced Local Bus Controller—Block Base Address 0x0_5000				
Offset	Register	Access	Reset	Section/Page
0x000	BR0—Base register 0	R/W	0x0000_nnnn	11.3.1.1/11-9
0x008	BR1—Base register 1	R/W	All zeros	11.3.1.1/11-9
0x010	BR2—Base register 2	R/W	All zeros	11.3.1.1/11-9
0x018	BR3—Base register 3	R/W	All zeros	11.3.1.1/11-9
0x020	BR4—Base register 4	R/W	All zeros	11.3.1.1/11-9
0x028	BR5—Base register 5	R/W	All zeros	11.3.1.1/11-9
0x030	BR6—Base register 6	R/W	All zeros	11.3.1.1/11-9
0x038	BR7—Base register 7	R/W	All zeros	11.3.1.1/11-9
0x004	OR0—Options register 0	R/W	0x0000_0FF7	11.3.1.2/11-11
0x00C	OR1—Options register 1	R/W	All zeros	11.3.1.2/11-11
0x014	OR2—Options register 2	R/W	All zeros	11.3.1.2/11-11
0x01C	OR3—Options register 3	R/W	All zeros	11.3.1.2/11-11
0x024	OR4—Options register 4	R/W	All zeros	11.3.1.2/11-11
0x02C	OR5—Options register 5	R/W	All zeros	11.3.1.2/11-11
0x034	OR6—Options register 6	R/W	All zeros	11.3.1.2/11-11
0x03C	OR7—Options register 7	R/W	All zeros	11.3.1.2/11-11
0x068	MAR—UPM address register	R/W	All zeros	11.3.1.3/11-19
0x06C	Reserved	—	—	—
0x070	MAMR—UPMA mode register	R/W	All zeros	11.3.1.4/11-20
0x074	MBMR—UPMB mode register	R/W	All zeros	11.3.1.4/11-20
0x078	MCMR—UPMC mode register	R/W	All zeros	11.3.1.4/11-20
0x07C–0x080	Reserved	—	—	—
0x084	MRTPR—Memory refresh timer prescaler register	R/W	All zeros	11.3.1.5/11-22
0x088	MDR—UPM/FCM data register	R/W	All zeros	11.3.1.6/11-22
0x08C	Reserved	—	—	—
0x090	LSOR—Special operation initiation register	R/W	All zeros	11.3.1.7/11-23
0x094–0x09C	Reserved	—	—	—
0x0A0	LURT—UPM refresh timer	R/W	All zeros	11.3.1.8/11-24

Table 11-3. Enhanced Local Bus Controller Registers (continued)

Enhanced Local Bus Controller—Block Base Address 0x0_5000				
Offset	Register	Access	Reset	Section/Page
0x0A4–0x0AC	Reserved	—	—	—
0x0B0	LTESR—Transfer error status register	w1c	All zeros	11.3.1.9/11-25
0x0B4	LTEDR—Transfer error disable register	R/W	All zeros	11.3.1.10/11-27
0x0B8	LTEIR—Transfer error interrupt register	R/W	All zeros	11.3.1.11/11-28
0x0BC	LTEATR—Transfer error attributes register	R/W	All zeros	11.3.1.12/11-29
0x0C0	LTEAR—Transfer error address register	R/W	All zeros	11.3.1.13/11-30
0x0C4–0x0CC	Reserved	—	—	—
0x0D0	LBCR—Configuration register	R/W	0x0004_0000	11.3.1.14/11-30
0x0D4	LCRR—Clock ratio register	R/W	0x8000_0008	11.3.1.15/11-32
0x0D8–0x0DC	Reserved	—	—	—
0x0E0	FMR—Flash mode register	R/W	0x0000_0n00	11.3.1.16/11-33
0x0E4	FIR—Flash instruction register	R/W	All zeros	11.3.1.17/11-34
0x0E8	FCR—Flash command register	R/W	All zeros	11.3.1.18/11-35
0x0EC	FBAR—Flash block address register	R/W	All zeros	11.3.1.19/11-36
0x0F0	FPAR—Flash page address register	R/W	All zeros	11.3.1.20/11-36
0x0F4	FBCR—Flash byte count register	R/W	All zeros	11.3.1.21/11-38
0x0F8–0x0FC	Reserved	—	—	—

11.3.1 Register Descriptions

This section provides a detailed description of the eLBC configuration, status, and control registers with detailed bit and field descriptions.

Address offsets in the eLBC address range that are not defined in [Table 11-3](#) 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.

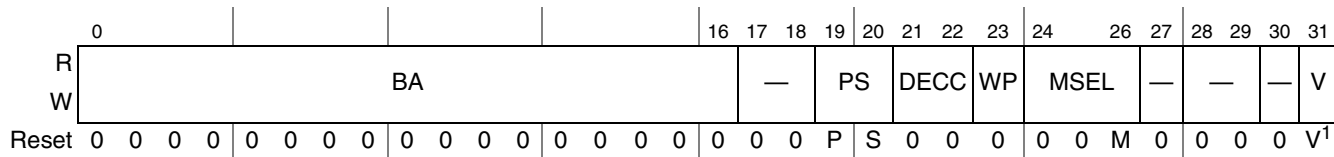
Bits designated as write-one-to-clear are cleared only by writing ones to them. Writing zeros to them has no effect.

11.3.1.1 Base Registers (BR0–BR3)

The base registers (BR n), shown in [Figure 11-2](#), 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]–BR3[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 BR0: 0x000 Access: Read/Write
 BR1: 0x008
 BR2: 0x010
 BR3: 0x018



¹ BR0 has its valid bit (V) set for RCWH[ROMLOC] = LBC. Thus bank 0 is valid with the port size (PS) configured from RCWH[ROMLOC] as loaded during reset. M = 0 for MSEL of GPCM, 1 for MSEL of FCM at boot. All other base registers have all bits cleared to zero during reset.

Figure 11-2. Base Registers (BR_n)

Table 11-4 describes BR_n fields.

Table 11-4. 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 the reset configuration word as loaded during reset. For all other banks the value is reset to 00 (port size not defined). 00 Reserved 01 8-bit (supported for GPCM, UPM, FCM) 10 16-bit (supported for GPCM, UPM) 11 Reserved
21–22	DECC	Specifies the method for data error checking. 00 Data error checking disabled. No ECC generation for FCM. 01 ECC checking is enabled, but ECC generation is disabled, for FCM on full-page transfers. 10 ECC checking and generation are enabled for FCM on full-page transfers. 11 Reserved
23	WP	Write protect. 0 Read and write accesses are allowed. 1 Only read accesses are allowed. The memory controller does not assert \overline{LCSn} on write cycles to this memory bank. LTESR[WP] is set (if WP is set) if a write to this memory bank is attempted, and a local bus error interrupt is generated (if enabled), terminating the cycle.

Table 11-4. BR_n Field Descriptions (continued)

Bits	Name	Description
24–26	MSEL	Machine select. Specifies the machine to use for handling memory operations. 000 GPCM (possible reset value) 001 FCM (possible reset value) 010 Reserved 011 Reserved 100 UPMA 101 UPMB 110 UPMC 111 Reserved
27	—	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{LCSn} does not assert unless V is set (an access to a region that has no valid bit set may cause a bus time-out). After a system reset, only BR0[V] is set. 0 This bank is invalid. 1 This bank is valid.

11.3.1.2 Option Registers (OR0–OR3)

The OR_n registers define the sizes of memory banks and access attributes. The OR_n attribute bits support the following three modes of operation as defined by BR_n[MSEL]:

- GPCM mode
- FCM mode
- UPM mode

The OR_n registers are interpreted differently depending on which of the three machine types is selected for that bank. Because bank 0 can be used to boot, the reset value of OR0 may be different depending on power-on configuration options. [Table 11-5](#) shows the reset values for OR0.

Table 11-5. Reset value of OR0 Register

Boot Source	OR0 Reset Value
FCM (small page NAND Flash)	0000_03AE
FCM (large page NAND Flash)	0000_07AE
GPCM	0000_0FF7
eLBC not used as a boot source	0000_0F07

11.3.1.2.1 Address Mask

The address mask field of the option registers (OR_n[AM]) masks up to 17 corresponding BR_n[BA] fields. The 15 LSBs of the 32-bit internal transaction address do not participate in bank address matching in selecting a bank for access. Masking address bits independently allows external devices of different size address ranges to be used. Address mask bits can be set or cleared in any order in the field, allowing a

resource to reside in more than one area of the address map. [Table 11-6](#) shows memory bank sizes from 32 Kbytes to 4 Gbytes. Memory block sizes vary from 32 Kbytes to 4 Gbytes in FCM mode, and 32 Kbytes to 64 Mbytes in GPCM and UPM modes.

Table 11-6. Memory Bank Sizes in Relation to Address Mask

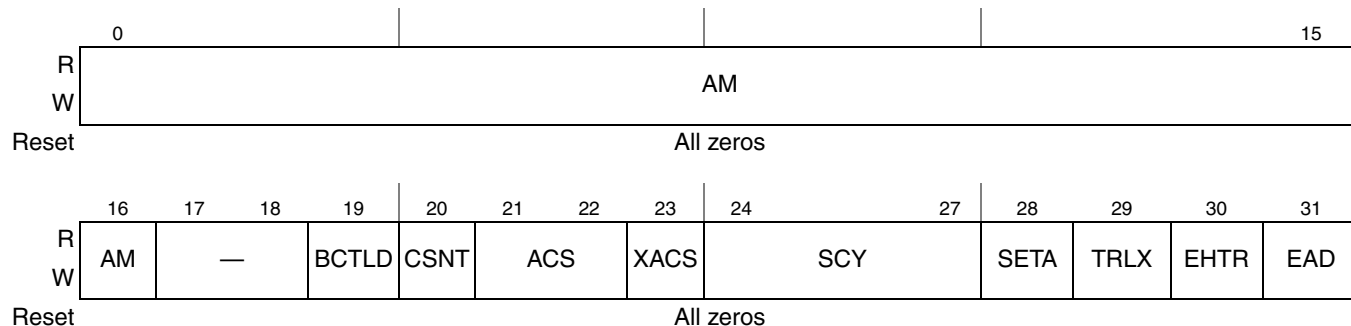
AM	Memory Bank Size
0000_0000_0000_0000_0	4 Gbytes
1000_0000_0000_0000_0	2 Gbytes
1100_0000_0000_0000_0	1 Gbyte
1110_0000_0000_0000_0	512 Mbytes
1111_0000_0000_0000_0	256 Mbytes
1111_1000_0000_0000_0	128 Mbytes
1111_1100_0000_0000_0	64 Mbytes
1111_1110_0000_0000_0	32 Mbytes
1111_1111_0000_0000_0	16 Mbytes
1111_1111_1000_0000_0	8 Mbytes
1111_1111_1100_0000_0	4 Mbytes
1111_1111_1110_0000_0	2 Mbytes
1111_1111_1111_0000_0	1 Mbyte
1111_1111_1111_1000_0	512 Kbytes
1111_1111_1111_1100_0	256 Kbytes
1111_1111_1111_1110_0	128 Kbytes
1111_1111_1111_1111_0	64 Kbytes
1111_1111_1111_1111_1	32 Kbytes

11.3.1.2.2 Option Registers (OR_n)—GPCM Mode

Figure 11-3 shows the bit fields for OR_n when the corresponding BR_n[MSEL] selects the GPCM machine.

Offset OR0: 0x004
 OR1: 0x00C
 OR2: 0x014
 OR3: 0x01C

Access: Read/Write



¹ Refer to Table 11-5 for the OR0 reset value. All other option registers have all bits cleared.

Figure 11-3. Option Registers (OR_n) in GPCM Mode

Table 11-7 describes OR_n fields for GPCM mode.

Table 11-7. OR_n—GPCM Field Descriptions

Bits	Name	Description
0–16	AM	GPCM address mask. Masks corresponding BR _n bits. Masking address bits independently allows external devices of different size address ranges to be used. Address mask bits can be set or cleared in any order in the field, allowing a resource to reside in more than one area of the address map. 0 Corresponding address bits are masked and therefore don't care for address checking. 1 Corresponding address bits are used in the comparison between base and transaction addresses.
17–18	—	Reserved
19	BCTLD	Buffer control disable. Disables assertion of LBCTL during access to the current memory bank. 0 LBCTL is asserted upon access to the current memory bank. 1 LBCTL is not asserted upon access to the current memory bank.
20	CSNT	Chip select negation time. Determines when \overline{LCSn} and \overline{LWE} are negated during an external memory write access handled by the GPCM, provided that ACS ≠ 00 (when ACS = 00, only \overline{LWE} is affected by the setting of CSNT). This helps meet address/data hold times for slow memories and peripherals. 0 \overline{LCSn} and \overline{LWE} are negated normally. 1 \overline{LCSn} and \overline{LWE} are negated earlier depending on the value of LCRR[CLKDIV].

LCRR [CLKDIV]	CSNT	Meaning
x	0	\overline{LCSn} and \overline{LWE} are negated normally.
2	1	\overline{LCSn} and \overline{LWE} are negated normally.
4 or 8	1	\overline{LCSn} and \overline{LWE} are negated one quarter bus clock cycle earlier.

Table 11-7. OR_n—GPCM Field Descriptions (continued)

Bits	Name	Description																		
21–22	ACS	<p>Address to chip-select setup. Determines the delay of the \overline{LCSn} assertion relative to the address change when the external memory access is handled by the GPCM. At system reset, OR0[ACS] = 11.</p> <table border="1" style="margin-left: 40px;"> <thead> <tr> <th>LCRR [CLKDIV]</th> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td rowspan="2">x</td> <td>00</td> <td>\overline{LCSn} is output at the same time as the address lines. Note that this overrides the value of CSNT such that CSNT = 0.</td> </tr> <tr> <td>01</td> <td>Reserved.</td> </tr> <tr> <td rowspan="2">2</td> <td>10</td> <td>\overline{LCSn} is output one half bus clock cycle after the address lines.</td> </tr> <tr> <td>11</td> <td>\overline{LCSn} is output one half bus clock cycle after the address lines.</td> </tr> <tr> <td rowspan="2">4 or 8</td> <td>10</td> <td>\overline{LCSn} is output one quarter bus clock cycle after the address lines.</td> </tr> <tr> <td>11</td> <td>\overline{LCSn} is output one half bus clock cycle after the address lines.</td> </tr> </tbody> </table>	LCRR [CLKDIV]	Value	Meaning	x	00	\overline{LCSn} is output at the same time as the address lines. Note that this overrides the value of CSNT such that CSNT = 0.	01	Reserved.	2	10	\overline{LCSn} is output one half bus clock cycle after the address lines.	11	\overline{LCSn} is output one half bus clock cycle after the address lines.	4 or 8	10	\overline{LCSn} is output one quarter bus clock cycle after the address lines.	11	\overline{LCSn} is output one half bus clock cycle after the address lines.
LCRR [CLKDIV]	Value	Meaning																		
x	00	\overline{LCSn} is output at the same time as the address lines. Note that this overrides the value of CSNT such that CSNT = 0.																		
	01	Reserved.																		
2	10	\overline{LCSn} is output one half bus clock cycle after the address lines.																		
	11	\overline{LCSn} is output one half bus clock cycle after the address lines.																		
4 or 8	10	\overline{LCSn} is output one quarter bus clock cycle after the address lines.																		
	11	\overline{LCSn} is output one half bus clock cycle after the address lines.																		
23	XACS	<p>Extra address to chip-select setup. Setting this bit increases the delay of the \overline{LCSn} assertion relative to the address change when the external memory access is handled by the GPCM. After a system reset, OR0[XACS] = 1.</p> <p>0 Address to chip-select setup is determined by ORx[ACS] and LCRR[CLKDIV]. 1 Address to chip-select setup is extended (see Table 11-30 and Table 11-31).</p>																		
24–27	SCY	<p>Cycle length in bus clocks. Determines the number of wait states inserted in the bus cycle, when the GPCM handles the external memory access. Thus it is the main parameter for determining cycle length. The total cycle length depends on other timing attribute settings. After a system reset, OR0[SCY] = 1111.</p> <p>0000 No wait states 0001 1 bus clock cycle wait state ... 1111 15 bus clock cycle wait states</p>																		
28	SETA	<p>External address termination.</p> <p>0 Access is terminated internally by the memory controller unless the external device asserts \overline{LGTA} earlier to terminate the access. 1 Access is terminated externally by asserting the \overline{LGTA} external pin. (Only \overline{LGTA} can terminate the access).</p>																		
29	TRLX	<p>Timing relaxed. Modifies the settings of timing parameters for slow memories or peripherals.</p> <p>0 Normal timing is generated by the GPCM. 1 Relaxed timing on the following parameters:</p> <ul style="list-style-type: none"> • 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{LCSn} (only if ACS is not equal to 00) and \overline{LWE} signals are negated one cycle earlier during writes. 																		

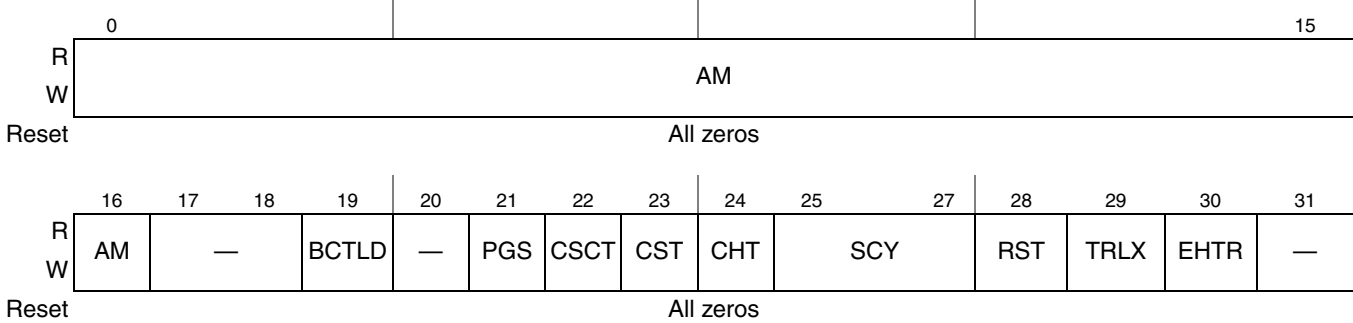
Table 11-7. OR_n—GPCM Field Descriptions (continued)

Bits	Name	Description															
30	EHTR	Extended hold time on read accesses. Indicates with TRLX how many cycles are inserted between a read access from the current bank and the next access. <table border="1" style="margin-left: 20px;"> <thead> <tr> <th>TRLX</th> <th>EHTR</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>The memory controller generates normal timing. No additional cycles are inserted.</td> </tr> <tr> <td>0</td> <td>1</td> <td>1 idle clock cycle is inserted.</td> </tr> <tr> <td>1</td> <td>0</td> <td>4 idle clock cycles are inserted.</td> </tr> <tr> <td>1</td> <td>1</td> <td>8 idle clock cycles are inserted.</td> </tr> </tbody> </table>	TRLX	EHTR	Meaning	0	0	The memory controller generates normal timing. No additional cycles are inserted.	0	1	1 idle clock cycle is inserted.	1	0	4 idle clock cycles are inserted.	1	1	8 idle clock cycles are inserted.
TRLX	EHTR	Meaning															
0	0	The memory controller generates normal timing. No additional cycles are inserted.															
0	1	1 idle clock cycle is inserted.															
1	0	4 idle clock cycles are inserted.															
1	1	8 idle clock cycles are inserted.															
31	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]).															

11.3.1.2.3 Option Registers (OR_n)—FCM Mode

Figure 11-4 shows the bit fields for OR_n when the corresponding BR_n[MSEL] selects the FCM machine.

Offset OR0: 0x004 OR4: 0x024 Access: Read/Write
 OR1: 0x00C OR5: 0x02C
 OR2: 0x014 OR6: 0x034
 OR3: 0x01C OR7: 0x03C



¹ Refer to Table 11-5 for the OR0 reset value. All other option registers have all bits cleared.

Figure 11-4. Option Registers (OR_n) in FCM Mode

Table 11-8 describes OR_n fields for FCM mode.

Table 11-8. OR_n—FCM Field Descriptions

Bits	Name	Description															
0–16	AM	FCM 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.															
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	—	Reserved															
21	PGS	NAND Flash EEPROM page size, buffer size, and block size. 0 Page size of 512 main area bytes plus 16 spare area bytes (small page devices); FCM RAM buffers are 1 Kbyte each; Flash block size of 16 Kbytes. 1 Page size of 2048 main area bytes plus 64 spare area bytes (large page devices); FCM RAM buffers are 4 Kbytes each; Flash block size of 128 Kbytes.															
22	CSCT	Chip select to command time. Determines how far in advance \overline{LCSn} is asserted prior to any bus activity during a NAND Flash access handled by the FCM. This helps meet chip-select setup times for slow memories. <table border="1" data-bbox="391 963 1442 1203"> <thead> <tr> <th>TRLX</th> <th>CSCT</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>The chip-select is asserted 1 clock cycle before any command.</td> </tr> <tr> <td>0</td> <td>1</td> <td>The chip-select is asserted 4 clock cycles before any command.</td> </tr> <tr> <td>1</td> <td>0</td> <td>The chip-select is asserted 2 clock cycles before any command.</td> </tr> <tr> <td>1</td> <td>1</td> <td>The chip-select is asserted 8 clock cycles before any command.</td> </tr> </tbody> </table>	TRLX	CSCT	Meaning	0	0	The chip-select is asserted 1 clock cycle before any command.	0	1	The chip-select is asserted 4 clock cycles before any command.	1	0	The chip-select is asserted 2 clock cycles before any command.	1	1	The chip-select is asserted 8 clock cycles before any command.
TRLX	CSCT	Meaning															
0	0	The chip-select is asserted 1 clock cycle before any command.															
0	1	The chip-select is asserted 4 clock cycles before any command.															
1	0	The chip-select is asserted 2 clock cycles before any command.															
1	1	The chip-select is asserted 8 clock cycles before any command.															
23	CST	Command setup time. Determines the delay of \overline{LFWEn} assertion relative to the command, address, or data change when the external memory access is handled by the FCM. <table border="1" data-bbox="391 1310 1442 1606"> <thead> <tr> <th>TRLX</th> <th>CST</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>The write-enable is asserted coincident with any command.</td> </tr> <tr> <td>0</td> <td>1</td> <td>The write-enable is asserted 0.25 clock cycles after any command, address, or data.</td> </tr> <tr> <td>1</td> <td>0</td> <td>The write-enable is asserted 0.5 clock cycles after any command, address, or data.</td> </tr> <tr> <td>1</td> <td>1</td> <td>The write-enable is asserted 1 clock cycle after any command, address, or data.</td> </tr> </tbody> </table>	TRLX	CST	Meaning	0	0	The write-enable is asserted coincident with any command.	0	1	The write-enable is asserted 0.25 clock cycles after any command, address, or data.	1	0	The write-enable is asserted 0.5 clock cycles after any command, address, or data.	1	1	The write-enable is asserted 1 clock cycle after any command, address, or data.
TRLX	CST	Meaning															
0	0	The write-enable is asserted coincident with any command.															
0	1	The write-enable is asserted 0.25 clock cycles after any command, address, or data.															
1	0	The write-enable is asserted 0.5 clock cycles after any command, address, or data.															
1	1	The write-enable is asserted 1 clock cycle after any command, address, or data.															

Table 11-8. OR_n—FCM Field Descriptions (continued)

Bits	Name	Description															
24	CHT	<p>Command hold time. Determines the $\overline{\text{LFWE}}$ negation prior to the command, address, or data change when the external memory access is handled by the FCM.</p> <table border="1"> <thead> <tr> <th>TRLX</th> <th>CHT</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>The write-enable is negated 0.5 clock cycles before any command, address, or data change.</td> </tr> <tr> <td>0</td> <td>1</td> <td>The write-enable is negated 1 clock cycle before any command, address, or data change.</td> </tr> <tr> <td>1</td> <td>0</td> <td>The write-enable is negated 1.5 clock cycles before any command, address, or data change.</td> </tr> <tr> <td>1</td> <td>1</td> <td>The write-enable is negated 2 clock cycles before any command, address, or data change.</td> </tr> </tbody> </table>	TRLX	CHT	Meaning	0	0	The write-enable is negated 0.5 clock cycles before any command, address, or data change.	0	1	The write-enable is negated 1 clock cycle before any command, address, or data change.	1	0	The write-enable is negated 1.5 clock cycles before any command, address, or data change.	1	1	The write-enable is negated 2 clock cycles before any command, address, or data change.
TRLX	CHT	Meaning															
0	0	The write-enable is negated 0.5 clock cycles before any command, address, or data change.															
0	1	The write-enable is negated 1 clock cycle before any command, address, or data change.															
1	0	The write-enable is negated 1.5 clock cycles before any command, address, or data change.															
1	1	The write-enable is negated 2 clock cycles before any command, address, or data change.															
25–27	SCY	<p>Cycle length in bus clocks. Determines the following:</p> <ul style="list-style-type: none"> the number of wait states inserted in command, address, or data transfer bus cycles, when the FCM handles the external memory access. Thus it is the main parameter for determining cycle length. The total cycle length depends on other timing attribute settings. the delay between command/address writes and data write cycles, or the delay between write cycles and read cycles from NAND Flash EEPROM. A delay of $4 \times (2 + \text{SCY})$ clock cycles (TRLX = 0) or $8 \times (2 + \text{SCY})$ clock cycles (TRLX = 1) is inserted between the last write and the first data transfer to/from NAND Flash devices. the delay between a command write and the first sample point of the RDY/$\overline{\text{BSY}}$ pin (connected to $\overline{\text{LFRB}}$). $\overline{\text{LFRB}}$ is not sampled until $8 \times (2 + \text{SCY})$ clock cycles (TRLX = 0) or $16 \times (2 + \text{SCY})$ clock cycles (TRLX = 1) have elapsed following the command. <p>000 No extra wait states 001 1 bus clock cycle wait state ... 111 7 bus clock cycle wait states</p>															
28	RST	<p>Read setup time. Determines the delay of $\overline{\text{LFRE}}$ assertion relative to sampling of read data when the external memory access is handled by the FCM.</p> <table border="1"> <thead> <tr> <th>TRLX</th> <th>RST</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>The read-enable is asserted 0.75 clock cycles prior to any wait states.</td> </tr> <tr> <td>0</td> <td>1</td> <td>The read-enable is asserted 1 clock cycle prior to any wait states.</td> </tr> <tr> <td>1</td> <td>0</td> <td>The read-enable is asserted 0.5 clock cycles prior to any wait states.</td> </tr> <tr> <td>1</td> <td>1</td> <td>The read-enable is asserted 1 clock cycle prior to any wait states.</td> </tr> </tbody> </table>	TRLX	RST	Meaning	0	0	The read-enable is asserted 0.75 clock cycles prior to any wait states.	0	1	The read-enable is asserted 1 clock cycle prior to any wait states.	1	0	The read-enable is asserted 0.5 clock cycles prior to any wait states.	1	1	The read-enable is asserted 1 clock cycle prior to any wait states.
TRLX	RST	Meaning															
0	0	The read-enable is asserted 0.75 clock cycles prior to any wait states.															
0	1	The read-enable is asserted 1 clock cycle prior to any wait states.															
1	0	The read-enable is asserted 0.5 clock cycles prior to any wait states.															
1	1	The read-enable is asserted 1 clock cycle prior to any wait states.															
29	TRLX	<p>Timing relaxed. Modifies the settings of timing parameters for slow memories.</p> <p>0 Normal timing is generated by the FCM.</p> <p>1 Relaxed timing on the following parameters:</p> <ul style="list-style-type: none"> Doubles the number of clock cycles between $\overline{\text{LCSn}}$ assertion and commands. Doubles the number of wait states specified by SCY, providing up to 14 wait states. Works in conjunction with CST and RST to extend command/address/data setup times. Adds one clock cycle to the command/address/data hold times. Works in conjunction with CBT to extend the wait time for read/busy status sampling by 16 clock cycles. Works in conjunction with EHTR to double hold time on read accesses. 															

Table 11-8. OR_n—FCM Field Descriptions (continued)

Bits	Name	Description															
30	EHTR	Extended hold time on read accesses. Indicates with TRLX how many cycles are inserted between a read access from the current bank and the next access.															
		<table border="1"> <thead> <tr> <th>TRLX</th> <th>EHTR</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1 idle clock cycle is inserted.</td> </tr> <tr> <td>0</td> <td>1</td> <td>2 idle clock cycles are inserted.</td> </tr> <tr> <td>1</td> <td>0</td> <td>4 idle clock cycles are inserted.</td> </tr> <tr> <td>1</td> <td>1</td> <td>8 idle clock cycles are inserted.</td> </tr> </tbody> </table>	TRLX	EHTR	Meaning	0	0	1 idle clock cycle is inserted.	0	1	2 idle clock cycles are inserted.	1	0	4 idle clock cycles are inserted.	1	1	8 idle clock cycles are inserted.
		TRLX	EHTR	Meaning													
		0	0	1 idle clock cycle is inserted.													
		0	1	2 idle clock cycles are inserted.													
1	0	4 idle clock cycles are inserted.															
1	1	8 idle clock cycles are inserted.															
31	—	Reserved															

11.3.1.2.4 Option Registers (OR_n)—UPM Mode

Figure 11-5 shows the bit fields for OR_n when the corresponding BR_n[MSEL] selects a UPM machine.

Offset OR0: 0x004 OR4: 0x024 Access: Read/Write
 OR1: 0x00C OR5: 0x02C
 OR2: 0x014 OR6: 0x034
 OR3: 0x01C OR7: 0x03C



¹ Refer to Table 11-5 for the OR0 reset value. All other option registers have all bits cleared.

Figure 11-5. Option Registers (OR_n) in UPM Mode

Table 11-9 describes BR_n fields for UPM mode.

Table 11-9. 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 pins.
17–18	—	Reserved
19	BCTLD	Buffer control disable. Disables assertion of LBCTL during access to the current memory bank. 0 LBCTL is asserted upon access to the current memory bank. 1 LBCTL is not asserted upon access to the current memory bank.

Table 11-9. ORn—UPM Field Descriptions (continued)

Bits	Name	Description																
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.																
30	EHTR	Extended hold time on read accesses. Indicates with TRLX how many cycles are inserted between a read access from the current bank and the next access.																
<table border="1"> <thead> <tr> <th>TRLX</th> <th>EHTR</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>The memory controller generates normal timing. No additional cycles are inserted.</td> </tr> <tr> <td>0</td> <td>1</td> <td>1 idle clock cycle is inserted.</td> </tr> <tr> <td>1</td> <td>0</td> <td>4 idle clock cycles are inserted.</td> </tr> <tr> <td>1</td> <td>1</td> <td>8 idle clock cycles are inserted.</td> </tr> </tbody> </table>			TRLX	EHTR	Meaning	0	0	The memory controller generates normal timing. No additional cycles are inserted.	0	1	1 idle clock cycle is inserted.	1	0	4 idle clock cycles are inserted.	1	1	8 idle clock cycles are inserted.	
TRLX			EHTR	Meaning														
0			0	The memory controller generates normal timing. No additional cycles are inserted.														
0			1	1 idle clock cycle is inserted.														
1	0	4 idle clock cycles are inserted.																
1	1	8 idle clock cycles are inserted.																
31	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]).																

11.3.1.3 UPM Memory Address Register (MAR)

Figure 11-6 shows the fields of the UPM memory address register (MAR).



Figure 11-6. UPM Memory Address Register (MAR)

Table 11-10 describes the MAR fields.

Table 11-10. MAR Field Descriptions

Bits	Name	Description
0–31	A	Address that can be output to the address signals under control of the AMX bits in the UPM RAM word.

11.3.1.4 UPM Mode Registers (MxMR)

The UPM machine mode registers (MAMR, MBMR, and MCMR), shown in [Figure 11-7](#), contain the configuration for the three UPMs.

Offset MAMR: 0x070
 MBMR: 0x074
 MCMR: 0x078

Access: Read/Write

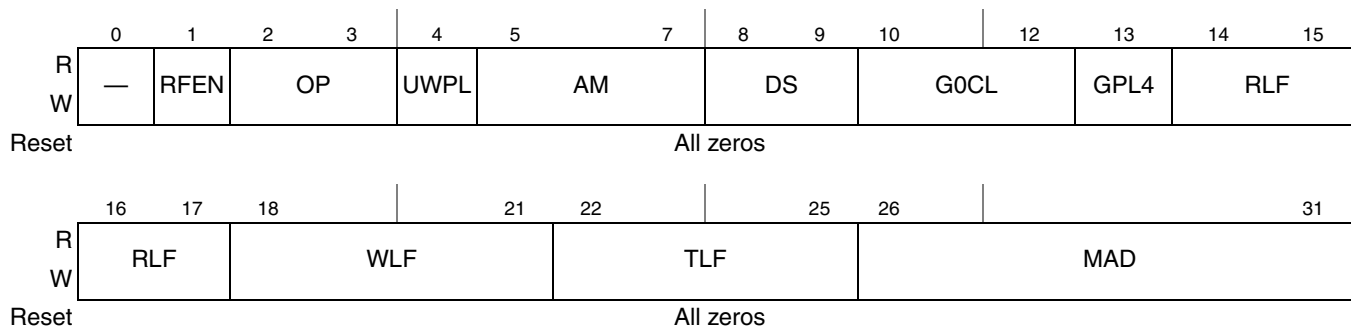


Figure 11-7. UPM Mode Registers (MxMR)

[Table 11-11](#) describes UPM mode fields.

Table 11-11. MxMR Field Descriptions

Bits	Name	Description
0	—	Reserved
1	RFEN	Refresh enable. Indicates that the UPM needs refresh services. This bit must be set for UPMA (refresh executor) if refresh services are required on any UPM assigned chip selects. If MAMR[RFEN] = 0, no refresh services can be provided, even if UPMB and/or UPMC have their RFEN bit set. 0 Refresh services are not required 1 Refresh services are required
2–3	OP	Command opcode. Determines the command executed by the UPM _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 pin when in UPM mode. 0 LUPWAIT is active high. 1 LUPWAIT is active low.

Table 11-11. MxMR Field Descriptions (continued)

Bits	Name	Description														
5–7	AM	<p>Address multiplex size. Determines how the address of the current memory cycle can be output on the address pins. This field is needed when interfacing with devices requiring row and column addresses multiplexed on the same pins. See Section 11.4.4.4.7, “Address Multiplexing (AMX)” for more information.</p> <p>000 Internal transaction address a[8:23] driven on [10:25]; LAD[0:15] driven low. 001 Internal transaction address a[7:22] driven on [10:25]; LAD[0:15] driven low. 010 Internal transaction address a[6:21] driven on [10:25]; LAD[0:15] driven low. 011 Internal transaction address a[5:20] driven on [10:25]; LAD[0:15] driven low. 100 Internal transaction address a[4:19] driven on [10:25]; LAD[0:15] driven low. 101 Internal transaction address a[3:18] driven on [10:25]; LAD[0:15] driven low. 110 Reserved 111 Reserved</p>														
8–9	DS	<p>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.</p> <p>00 1-bus clock cycle disable period 01 2-bus clock cycle disable period 10 3-bus clock cycle disable period 11 4-bus clock cycle disable period</p>														
10–12	G0CL	<p>General line 0 control. Determines which logical address line can be output to the LGPL0 pin when the UPMn is selected to control the memory access.</p> <p>000 A12 001 A11 010 A10 011 A9 100 A8 101 A7 110 A6 111 A5</p>														
13	GPL4	<p>LGPL4 output line disable. Determines how the LGPL4/LUPWAIT pin is controlled by the corresponding bits in the UPMn array. See Table 11-38.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th rowspan="2">Value</th> <th rowspan="2">LGPL4/LUPWAIT Pin Function</th> <th colspan="2">Interpretation of UPM Word Bits</th> </tr> <tr> <th>G4T1/DLT3</th> <th>G4T3/WAEN</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>LGPL4 (output)</td> <td>G4T1</td> <td>G4T3</td> </tr> <tr> <td>1</td> <td>LUPWAIT (input)</td> <td>DLT3</td> <td>WAEN</td> </tr> </tbody> </table>	Value	LGPL4/LUPWAIT Pin Function	Interpretation of UPM Word Bits		G4T1/DLT3	G4T3/WAEN	0	LGPL4 (output)	G4T1	G4T3	1	LUPWAIT (input)	DLT3	WAEN
Value	LGPL4/LUPWAIT Pin Function	Interpretation of UPM Word Bits														
		G4T1/DLT3	G4T3/WAEN													
0	LGPL4 (output)	G4T1	G4T3													
1	LUPWAIT (input)	DLT3	WAEN													
14–17	RLF	<p>Read loop field. Determines the number of times a loop defined in the UPMn will be executed for a burst- or single-beat read pattern or when MxMR[OP] = 11 (RUN command)</p> <p>0000 16 0001 1 0010 2 0011 3 ... 1110 14 1111 15</p>														

Table 11-11. MxMR Field Descriptions (continued)

Bits	Name	Description
18–21	WLF	Write loop field. Determines the number of times a loop defined in the UPM n will be executed for a burst- or single-beat write pattern. 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 will be executed for a refresh service pattern. 0000 16 0001 1 0010 2 0011 3 ... 1110 14 1111 15
26–31	MAD	Machine address. RAM address pointer for the command executed. This field is incremented by 1, each time the UPM is accessed and the OP field is set to WRITE or READ. Address range is 64 words per UPM n .

11.3.1.5 Memory Refresh Timer Prescaler Register (MRTPR)

The refresh timer prescaler register (MRTPR), shown in Figure 11-8, is used to divide the csbclock (or twice csb_clk (if RCWL[LBCM] is set) to provide the UPM refresh timers clock.

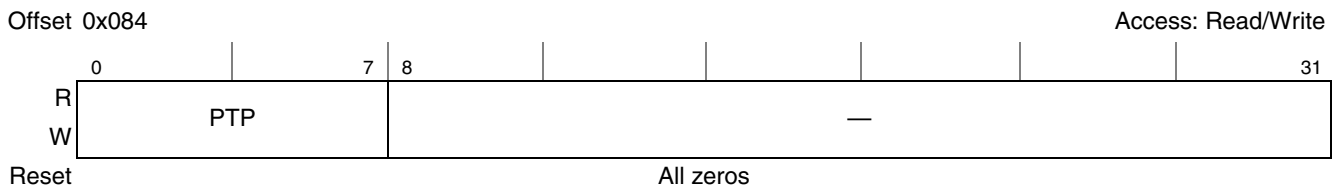


Figure 11-8. Memory Refresh Timer Prescaler Register (MRTPR)

Table 11-12 describes MRTPR fields.

Table 11-12. MRTPR Field Descriptions

Bits	Name	Description
0–7	PTP	Refresh timers prescaler. Determines the period of the refresh timers input clock. The csb clock (or twice csb_clk (if RCWL[LBCM] is set) is divided by PTP except when the value is 00000_0000, which represents the maximum divider of 256.
8–31	—	Reserved

11.3.1.6 UPM/FCM Data Register (MDR)

The memory data register (MDR), shown in Figure 11-9 and Figure 11-10, contains data written to or read from the RAM array for UPM read or write commands. MDR also contains data written to or read from

an external NAND Flash EEPROM for FCM write address, write data, and read status commands. MDR must be set up before issuing a write command to the UPM, or before issuing a FCM operation sequence that uses MDR to source address or data bytes.

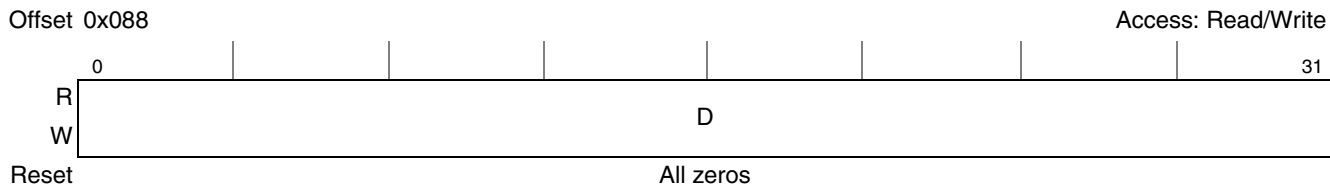


Figure 11-9. UPM Data Register in UPM Mode (MDR)



Figure 11-10. FCM Data Register in FCM Mode (MDR)

Table 11-13 describes MDR[D].

Table 11-13. MDR Field Description

Bits	Name	Description
0–31	D	In UPM mode, D is the data to be read or written into the RAM array when a write or read command is supplied to the UPM (MxMR[OP] = 01 or MxMR[OP] = 10).
0–7	AS3	In FCM mode, AS3 is the fourth byte of address sent by a custom address write operation, or the fourth byte of data read from a read status operation.
8–15	AS2	In FCM mode, AS2 is the third byte of address sent by a custom address write operation, or the third byte of data read from a read status operation.
16–23	AS1	In FCM mode, AS1 is the second byte of address sent by a custom address write operation, or the second byte of data read from a read status operation.
24–31	AS0	In FCM mode, AS0 is the first byte of address sent by a custom address write operation, or the first byte of data read from a read status operation.

11.3.1.7 Special Operation Initiation Register (LSOR)

The special operation initiation register (LSOR), shown in Figure 11-11, is used by software to trigger a special operation on the indicated bank. Writing to LSOR activates a special operation on bank LSOR[BANK] provided that the bank is valid and controlled by a memory controller whose mode OP field is set to a value other than “normal operation.” If eLBC is currently busy with a memory transaction, writing LSOR completes immediately, but the special operation request is queued until eLBC can service it. To avoid race conditions between software and a busy eLBC, registers that affect currently running special operation and LSOR must not be rewritten before a pending special operation has been completed. The UPM and FCM have different indications of when such special operations are completed. The

behavior of eLBC is unpredictable if special operation modes are altered between LSOR being written and the relevant memory controller completing that access.

UPM special operation modes are set in registers MxMR[OP], see [Section 11.3.1.4, “UPM Mode Registers \(MxMR\).”](#) FCM special operation modes are set in FMR[OP], see [Section 11.3.1.16, “Flash Mode Register \(FMR\).”](#) Writing LSOR has the same effect as setting a special controller mode and performing a dummy access to a bank associated with the controller in question, but use of LSOR avoids changing settings for the address space occupied by the bank. More details of special operation sequences appear in [Section 11.4.4.2.1, “UPM Programming Example \(Two Sequential Writes to the RAM Array\).”](#)

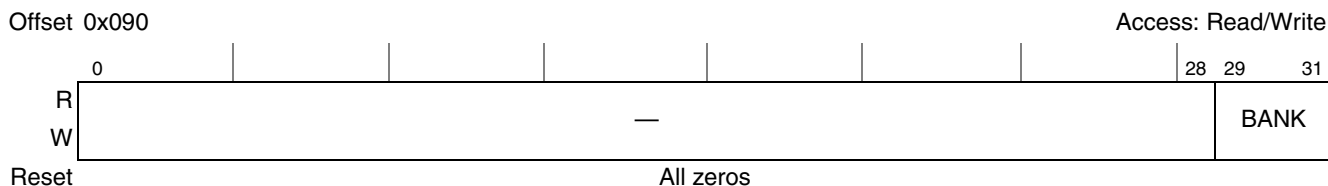


Figure 11-11. Special Operation Initiation Register (LSOR)

Table 11-14 describes LSOR.

Table 11-14. LSOR Field Description

Bits	Name	Description
0–28	—	Reserved
29–31	BANK	Bank on which a special operation is initiated. If the bank identified by BANK is marked valid (BRn[V] set) and the bank is controlled by a memory controller whose current mode OP is non-zero—or a special operation—eLBC will request the special operation to be activated on the selected bank when this field is written. Otherwise, writing this field has no effect. 000 Bank 0 is triggered for special operation ... 011 Bank 3 is triggered for special operation 100–110Reserved

11.3.1.8 UPM Refresh Timer (LURT)

The UPM refresh timer (LURT), shown in [Figure 11-12](#), generates a refresh request for all valid banks that selected a UPM machine and are refresh-enabled (MxMR[RFEN] = 1). Each time the timer expires, a qualified bank generates a refresh request using the selected UPM. The qualified banks rotate their requests.

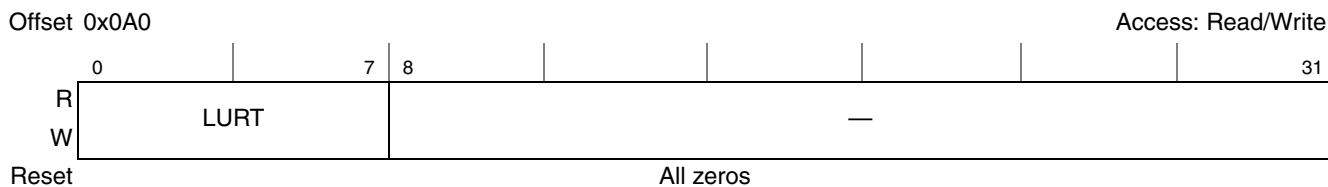


Figure 11-12. UPM Refresh Timer (LURT)

Table 11-15 describes LURT fields.

Table 11-15. LURT Field Descriptions

Bits	Name	Description
0–7	LURT	<p>UPM refresh timer period. Determines, along with the timer prescaler (MRTPR), the timer period according to the following equation:</p> $\text{TimerPeriod} = \frac{\text{LURT}}{\left(\frac{\text{Fsystemclock}}{\text{MRTPR[PTP]}}\right)}$ <p>Example: For a 266-MHz system clock and a required service rate of 15.6 μs, given MRTPR[PTP] = 32, the LURT value should be 128 decimal. 128/(266 MHz/32) = 15.4 μs, which is less than the required service period of 15.6 μs. Note that the reset value (0x00) sets the maximum period to 256 x MRTPR[PTP] system clock cycles.</p>
8–31	—	Reserved

11.3.1.9 Transfer Error Status Register (LTESR)

The transfer error status register (LTESR) indicates the cause of an error or event. LTESR, shown in Figure 11-13, is a write-1-to-clear register. Reading LTESR occurs normally; however, write operations can clear but not set bits. A bit is cleared whenever the register is written, and the data in the corresponding bit location is a 1. For example, to clear only the write protect error bit (LTESR[WP]) without affecting other LTESR bits, 0x0400_0000 should be written to the register. After any error/event reported by LTESR, LTEATR[V] must be cleared for LTESR to updated again.

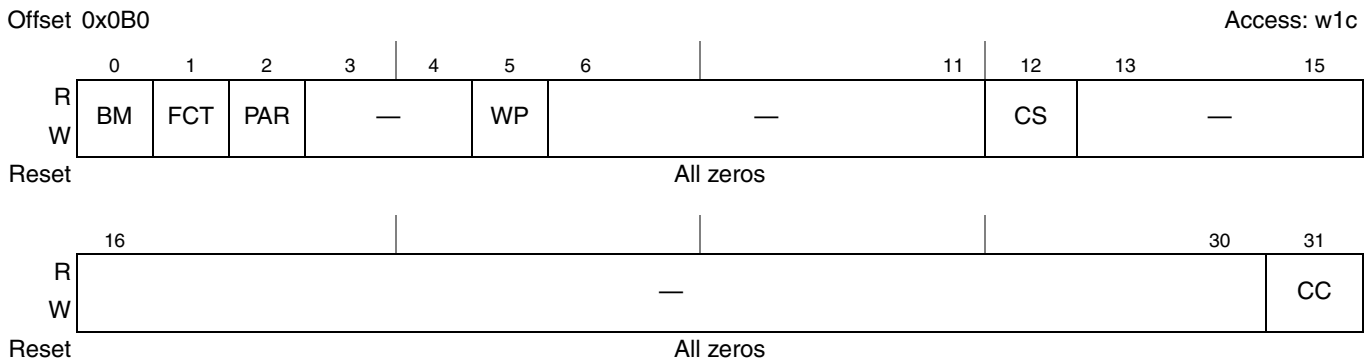


Figure 11-13. Transfer Error Status Register (LTESR)

Table 11-16 describes LTESR fields.

Table 11-16. LTESR Field Descriptions

Bits	Name	Description
0	BM	Bus monitor time-out 0 No local bus monitor time-out occurred. 1 Local bus monitor time-out occurred. No data beat was acknowledged on the bus within LBCR[BMT] x LBCR[BMTPS] bus clock cycles from the start of a transaction.
1	FCT	FCM command time-out 0 No FCM command time-out occurred. 1 A CW0, CW1, CW2, or CW3 command issued to FCM timed-out with respect to the timer configured by FMR[CWTO].
2	PAR	ECC error for FCM mode 0 No local bus ECC error 1 U ECC error (FCM). LTEATR[PB] indicates the block that caused the error and LTEATR[BNK] indicates which memory controller bank was accessed.
3–4	—	Reserved
5	WP	Write protect error 0 No write protect error occurred. 1 A write was attempted to a local bus memory region that was defined as read-only in the memory controller. Usually, in this case, a bus monitor time-out will occur (as the cycle is not automatically terminated).
6–11	—	Reserved
12	CS	Chip select error 0 No chip select error occurred. 1 A transaction was sent to the eLBC that did not hit any memory bank.
13–30	—	Reserved
31	CC	FCM command completion event 0 No FCM operation in progress, or operation pending. 1 FCM operation has completed, allowing software to continue processing of results.

11.3.1.10 Transfer Error Check Disable Register (LTEDR)

The transfer error check disable register (LTEDR), shown in [Figure 11-14](#), is used to disable error/event checking. Note that control of error/event checking is independent of control of reporting of errors/events (LTEIR) through the interrupt mechanism.

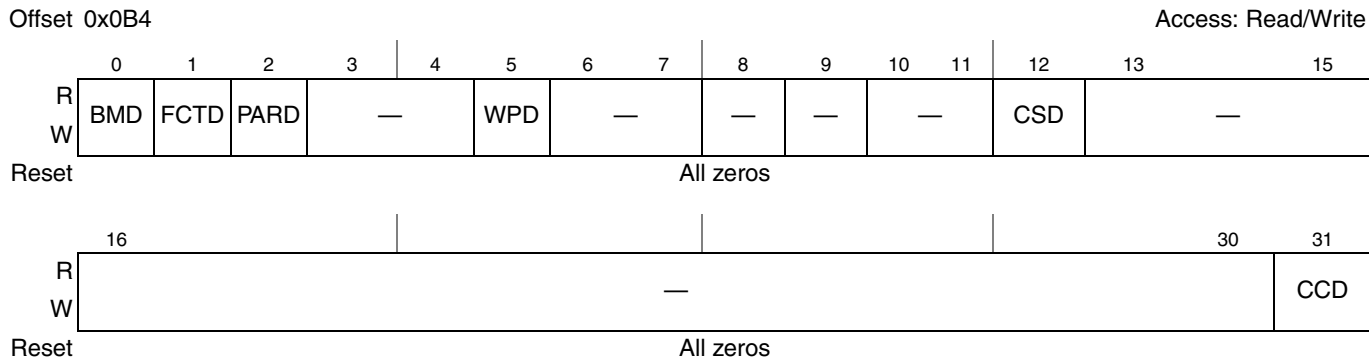


Figure 11-14. Transfer Error Check Disable Register (LTEDR)

[Table 11-17](#) describes LTEDR fields.

Table 11-17. LTEDR Field Descriptions

Bits	Name	Description
0	BMD	Bus monitor disable 0 Bus monitor is enabled. 1 Bus monitor is disabled, but internal bus time-outs can still occur.
1	FCTD	FCM command time-out disable 0 FCM command timer is enabled. 1 FCM command time-out is disabled, but internal FCM command timer can terminate command waits.
2	PARD	ECC error checking disabled. 0 ECC error checking is enabled. 1 ECC error checking is disabled.
3–4	—	Reserved
5	WPD	Write protect error checking disable. 0 Write protect error checking is enabled. 1 Write protect error checking is disabled.
6–11	—	Reserved
12	CSD	Chip select error checking disable. 0 Chip select error checking is enabled. 1 Chip select error checking is disabled.
13–30	—	Reserved
31	CCD	FCM command completion checking disable. 0 Command completion checking is enabled. 1 Command completion checking is disabled.

11.3.1.11 Transfer Error Interrupt Enable Register (LTEIR)

The transfer error interrupt enable register (LTEIR), shown in Figure 11-15, is used to send or block error/event reporting through the eLBC internal interrupt mechanism. Software should clear pending errors/events in LTESR before enabling interrupts. After an interrupt has occurred, clearing relevant LTESR error/event bits negates the interrupt.

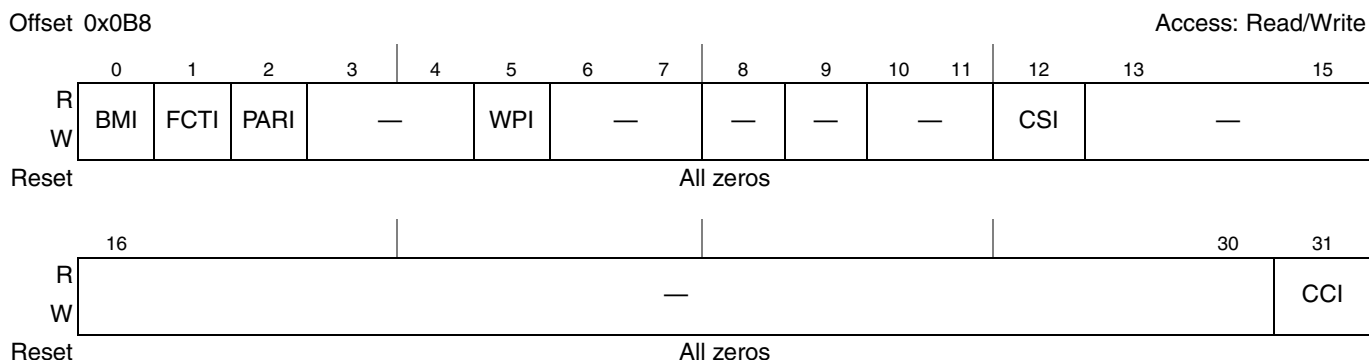


Figure 11-15. Transfer Error Interrupt Enable Register (LTEIR)

Table 11-18 describes LTEIR fields.

Table 11-18. LTEIR Field Descriptions

Bits	Name	Description
0	BMI	Bus monitor error interrupt enable. 0 Bus monitor error reporting is disabled. 1 Bus monitor error reporting is enabled.
1	FCTI	FCM command time-out interrupt enable. 0 FCM command time-out error reporting is disabled. 1 FCM command time-out error reporting is enabled.
2	PARI	ECC error interrupt enable. 0 ECC error reporting is disabled. 1 ECC error reporting is enabled.
3–4	—	Reserved
5	WPI	Write protect error interrupt enable. 0 Write protect error reporting is disabled. 1 Write protect error reporting is enabled.
6–11	—	Reserved
12	CSI	Chip select error interrupt enable. 0 Chip select error reporting is disabled. 1 Chip select error reporting is enabled.
13–30	—	Reserved
31	CCI	FCM command completion Event interrupt enable. 0 Command completion reporting is disabled. 1 Command completion reporting is enabled.

11.3.1.12 Transfer Error Attributes Register (LTEATR)

The transfer error attributes register (LTEATR) captures source attributes of an error/event. Figure 11-16 shows the LTEATR. After LTEATR[V] has been set, software must clear this bit to allow LTESR, LTEATR, and LTEAR to update following any subsequent events/errors.

NOTE

LTEATR may not capture accurate information for errors that occur when an FCM special operation is in progress.

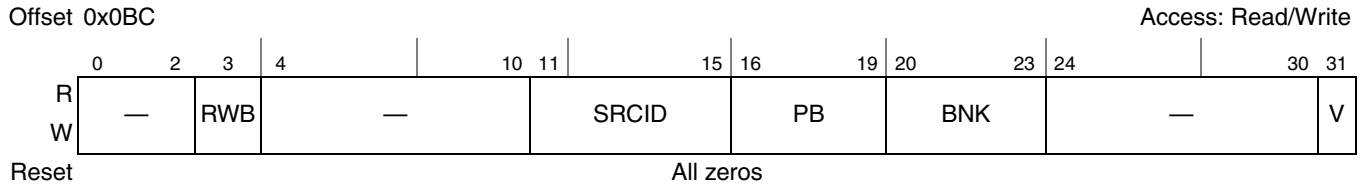


Figure 11-16. Transfer Error Attributes Register (LTEATR)

Table 11-19 describes LTEATR fields.

Table 11-19. LTEATR Field Descriptions

Bits	Name	Description
0–2	—	Reserved
3	RWB	Transaction type for the error: 0 The transaction for the error was a write transaction. 1 The transaction for the error was a read transaction.
4–10	—	Reserved
11–15	SRCID	Captures the source of the transaction when this information is provided on the internal interface to the eLBC. The coding of the source ID debug information is the same as the coding of AEATR[MSTR_ID] (see Section 7.2.6, “Arbiter Event Attributes Register (AEATR).”)
16–19	PB	Error on block for FCM. For FCM, there are at most four 512-byte page blocks (for a large page device) checked by ECC. A bit is set for the 512-byte block that had a non-correctable ECC error on read (bit 16 represents block 0, the first 512 bytes of a page; if ORx[PGS] = 0, bits 17–19 are always 0).
20–23	BNK	Memory controller bank. There is one error status bit per memory controller bank (bit 20 represents bank 0). A bit is set for the local bus memory controller bank that had an error.
24–30	—	Reserved
31	V	Error attribute capture is valid. Indicates that the captured error information is valid. 0 Captured error attributes and address are not valid. 1 Captured error attributes and address are valid.

11.3.1.13 Transfer Error Address Register (LTEAR)

The transfer error address register (LTEAR) captures the address of a transaction that caused an error/event. The transfer error address register (LTEAR) is shown in [Figure 11-17](#).

NOTE

LTEAR may not capture accurate information for errors that occur when an FCM special operation is in progress.

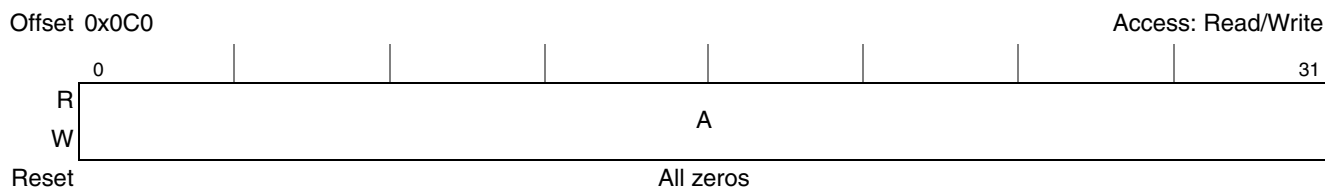


Figure 11-17. Transfer Error Address Register (LTEAR)

[Table 11-20](#) describes LTEAR fields.

Table 11-20. LTEAR Field Descriptions

Bits	Name	Description
0–31	A	Transaction address for the error. For GPCM and UPM, holds the 32-bit address of the transaction resulting in an error. For FCM, this register is undefined.

11.3.1.14 Local Bus Configuration Register (LBCR)

The local bus configuration register (LBCR) is shown in [Figure 11-18](#).

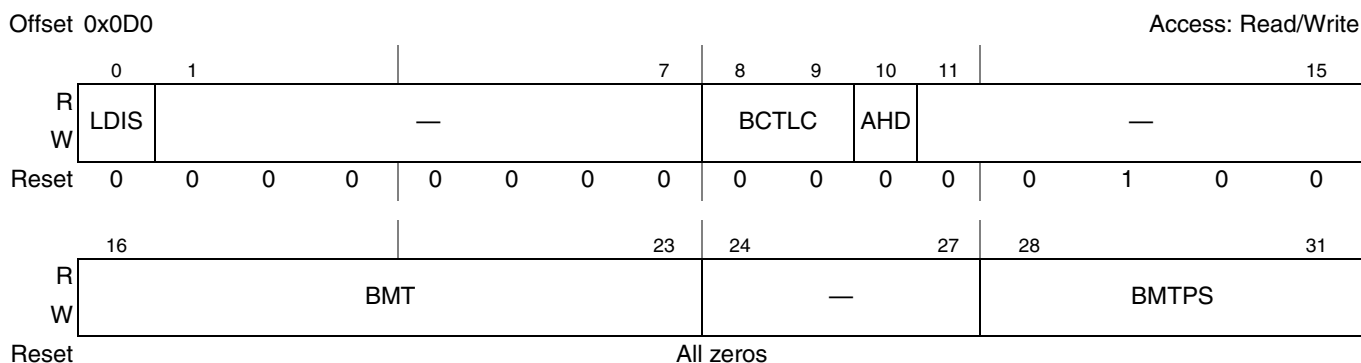


Figure 11-18. Local Bus Configuration Register

Table 11-21 describes LBCR fields.

Table 11-21. 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 $\overline{W/R}$ 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	AHD	Address hold disable. Removes part of the hold time for LAD with respect to LALE in order to lengthen the LALE pulse. 0 During address phases on the local bus, the LALE signal negates one platform clock period prior to the address being invalidated. 1 During address phases on the local bus, the LALE signal negates half platform clock period prior to the address being invalidated.
11–15	—	Reserved. Reads to bit 13 return 1.
16–23	BMT	Bus monitor timing. Defines the bus monitor time-out period. Clearing BMT (reset value) selects the maximum count of bus clock cycles. For non-zero values of BMT, the number of LCLK clock cycles to count down before a time-out error is generated is given by: bus cycles = BMT × PS, where PS is set according to LBCR[BMTPS]. The value of BMT × PS must not be less than 40 bus cycles for reliable operation.
24–27	—	Reserved
28–31	BMTPS	Bus monitor timer prescale. Defines the multiplier, PS, to scale LBCR[BMT] for determining bus time-outs. 0000 PS = 8 0001 PS = 16 0010 PS = 32 0011 PS = 64 0100 PS = 128 0101 PS = 256 0110 PS = 512 0111 PS = 1024 1000 PS = 2048 1001 PS = 4096 1010 PS = 8192 1011 PS = 16,384 1100 PS = 32,768 1101 PS = 65,536 1110 PS = 131,072 1111 PS = 262,144

11.3.1.15 Clock Ratio Register (LCRR)

The clock ratio register, shown in [Figure 11-19](#), sets the csb clock to eLBC bus frequency ratio. It also provides configuration bits for extra delay cycles for address and control signals.

NOTE

For proper operation of the system, it is required that this register setting will not be altered while local bus memories or devices are being accessed. Special care needs to be taken when running instructions from an eLBC memory.

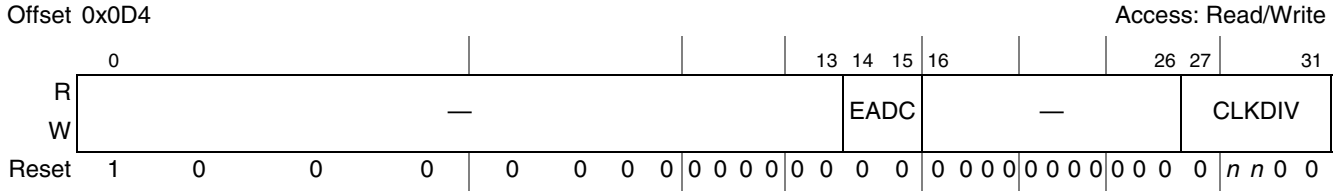


Figure 11-19. Clock Ratio Register (LCRR)

[Table 11-22](#) describes LCRR fields.

Table 11-22. LCRR Field Descriptions

Bits	Name	Description
0–13	—	Reserved. Note bit 0 must remain set for proper operation.
14–15	EADC	External address delay cycles of LCLK. Defines the number of cycles for the assertion of LALE. 00 4 01 1 10 2 11 3
16–26	—	Reserved
27–31	CLKDIV	Clock divider. Sets the frequency ratio between the csb clock (or twice csb_clk (if RCWL[LBCM] is set) and the local bus clock. Only the values shown below are allowed. Note: It is critical that no transactions are being executed via the local bus while CLKDIV is being modified. As such, prior to modification, the user must ensure that code is not executing out of the local bus. Once LCRR[CLKDIV] is written, the register should be read, and then an isync should be executed. 00000–00001 Reserved 00010 2 00011 Reserved 00100 4 00101–00111 Reserved 01000 8 01001–11111 Reserved

11.3.1.16 Flash Mode Register (FMR)

The local bus Flash mode register (FMR), shown in Figure 11-20, controls global operation of the FCM.

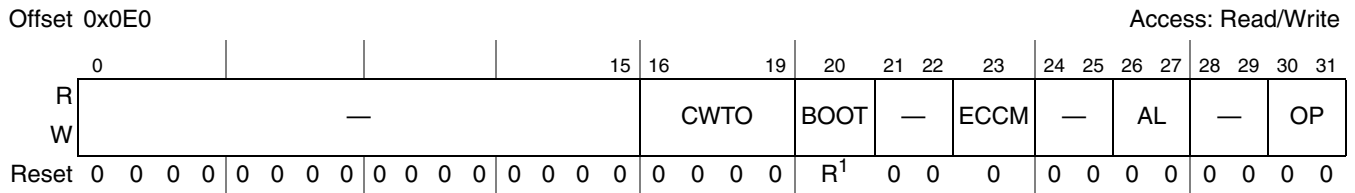


Figure 11-20. Flash Mode Register

¹ Bit R (field BOOT) is set if power-on-reset configuration selects FCM as the boot ROM target.

Table 11-23 describes FMR fields.

Table 11-23. FMR Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–19	CWTO	Command wait time-out. For FCM commands that wait on $\overline{\text{LFRB}}$ being sampled high (CW0, CW1, RBW, and RSW), FCM pauses execution of the instruction sequence until either $\overline{\text{LFRB}}$ is sampled high, or a timer controlled by CTO expires, whichever occurs first. The time-out in the latter case is as follows: 0000 256 cycles of LCLK 0001 512 cycles of LCLK 0010 1024 cycles of LCLK 0011 2048 cycles of LCLK 0100 4096 cycles of LCLK 0101 8192 cycles of LCLK 0110 16,384 cycles of LCLK 0111 32,768 cycles of LCLK 1000 65,536 cycles of LCLK 1001 131,072 cycles of LCLK 1010 262,144 cycles of LCLK 1011 524,288 cycles of LCLK 1100 1,048,576 cycles of LCLK 1101 2,097,152 cycles of LCLK 1110 4,194,304 cycles of LCLK 1111 8,388,608 cycles of LCLK
20	BOOT	Flash auto-boot load mode. During system boot from NAND Flash EEPROM, this bit remains set to alter the use of the FCM buffer RAM. Software should clear BOOT once FCM is to be restored to normal operation. Setting BOOT without auto-boot in progress only alters the mapping of the buffer RAM. 0 FCM is operating in normal functional mode, with an 8 Kbyte FCM buffer RAM. 1 eLBC has been configured—either from reset or by a special operation OP = 01—to autoload a 4-Kbyte boot block into the FCM buffer RAM, which maps only the 4 Kbytes of NAND Flash main data region comprising the boot block. Any access to the buffer RAM is delayed until the entire boot block has been loaded.
21–22	—	Reserved

Table 11-23. FMR Field Descriptions (continued)

Bits	Name	Description
23	ECCM	<p>ECC mode. When hardware checking and/or generation of error correcting codes (ECC) is enabled (that is, when BRn[DECC] is 01 or 10, and full page transfers are specified with FBCR[BC] = 0), ECCM sets the ECC block size and position of the ECC code word(s) in the NAND Flash spare region for both checking and generation functions. The format of the ECC code word conforms with the Samsung/Toshiba spare region assignment specifications.</p> <p>0 ECC is checked/calculated over 512-Byte blocks. A 24-bit ECC is assigned to spare region bytes at offsets $(N \times 16) + 6$ through $(N \times 16) + 8$ for spare region N, $N = 0-3$.</p> <p>1 ECC is checked/calculated over 512-Byte blocks. A 24-bit ECC is assigned to spare region bytes at offsets $(N \times 16) + 8$ through $(N \times 16) + 10$ for spare region N, $N = 0-3$.</p>
24–25	—	Reserved
26–27	AL	<p>Address length. AL sets the number of address bytes issued during page address (PA) operations. However, the number of address bytes issued for column address (CA) operations is determined by the device page size (for ORn[PGS] = 0, 1 CA byte is issued; for ORn[PGS] = 1, 2 CA bytes are issued).</p> <p>00 2 bytes are issued for page addresses, thus a total of 3 (ORn[PGS] = 0) or 4 (ORn[PGS] = 1) address bytes are issued for a {CA,PA} sequence</p> <p>01 3 bytes are issued for page addresses, thus a total of 4 (ORn[PGS] = 0) or 5 (ORn[PGS] = 1) address bytes are issued for a {CA,PA} sequence</p> <p>10 4 bytes are issued for page addresses, thus a total of 5 (ORn[PGS] = 0) or 6 (ORn[PGS] = 1) address bytes are issued for a {CA,PA} sequence</p> <p>11 —</p>
28–29	—	Reserved
30–31	OP	<p>Flash operation. For OP not equal to 00, a special operation is triggered on the next write to LSOR or dummy access to a bank controlled by FCM. Once a special operation has commenced, OP is automatically reset to 00 by FCM. Individual blocks may be temporarily unlocked for erase and reprogramming operations.</p> <p>00 Normal operation. All read and write accesses to banks controlled by FCM access the shared FCM buffer RAM. No bus activity is caused by this operation.</p> <p>01 Simulate auto-boot block loading, and set FMR[BOOT]. Boot block loading occurs from the bank triggered on the special operation, therefore the appropriate bank configuration must be initialized prior to issuing this operation.</p> <p>10 Execute the command sequence contained in FIR, but with write protection enabled (pin $\overline{\text{LFWP}}$ asserted low) so that all Flash blocks are protected from accidental erasure and reprogramming.</p> <p>11 Execute the command sequence contained in FIR, but permit the single block identified by FBAR[BLK] to be erased or reprogrammed, with pin $\overline{\text{LFWP}}$ remaining high during the access.</p>

11.3.1.17 Flash Instruction Register (FIR)

The local bus Flash instruction register (FIR), shown in [Figure 11-21](#), holds a sequence of up to eight instructions for issue by the FCM. Setting FMR[OP] non-zero and writing LSOR or accessing a bank controlled by FCM causes FCM to read FIR 4 bits at a time, starting at bit 0 and continuing with adjacent 4-bit opcodes, until only NOP opcodes remain. The programmed instruction sequence of OP0, OP1, ..., OP7 is performed on the activated bank, using the data buffer addressed by FPAR. If LTEIR[CCI] = 1 and LTEDR[CCD] = 0, eLBC will generate an interrupt once the entire sequence has completed, and software should examine LTEATR and clear its V bit.

Software must not alter the contents of the addressed FCM buffer, FIR, MDR, FCR, FBAR, FPAR, or FBCR while an operation is in progress—or eLBC will behave unpredictably—but software can freely modify the contents of any currently unused FCM RAM buffer in preparation for the next operation.

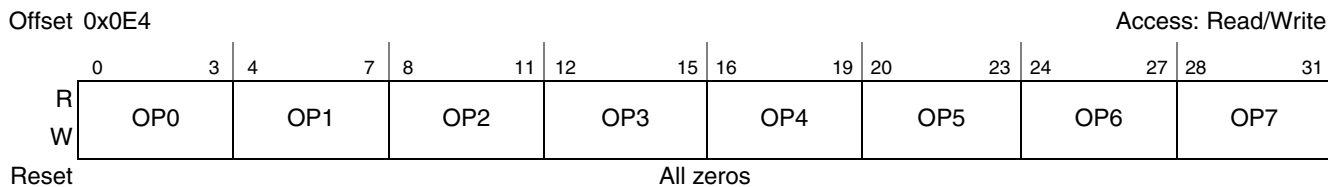


Figure 11-21. Flash Instruction Register

Table 11-24 describes FIR fields.

Table 11-24. FIR Field Descriptions

Bits	Name	Description
0–3	OP0	FCM operation codes. OP0 is executed first, followed by OP1, through to OP7.
4–7	OP1	0000 NOP—No-operation and end of operation sequence
		0001 CA—Issue current column address as set in FPAR, with length set by ORx[PGS]
8–11	OP2	0010 PA—Issue current block+page address as set in FBAR and FPAR, with length set by FMR[AL]
		0011 UA—Issue user-defined address byte from next AS field in MDR
12–15	OP3	0100 CM0—Issue command from FCR[CMD0]
		0101 CM1—Issue command from FCR[CMD1]
16–19	OP4	0110 CM2—Issue command from FCR[CMD2]
20–23	OP5	0111 CM3—Issue command from FCR[CMD3]
		1000 WB—Write FBCR bytes of data from current FCM buffer to Flash device
24–27	OP6	1001 WS—Write one byte (8b port) of data from next AS field of MDR to Flash device
		1010 RB—Read FBCR bytes of data from Flash device into current FCM RAM buffer
28–31	OP7	1011 RS—Read one byte (8b port) of data from Flash device into next AS field of MDR
		1100 CW0—Wait for $\overline{\text{LFRB}}$ to return high or time-out, then issue command from FCR[CMD0]
		1101 CW1—Wait for $\overline{\text{LFRB}}$ to return high or time-out, then issue command from FCR[CMD1]
		1110 RBW—Wait for $\overline{\text{LFRB}}$ to return high or time-out, then read FBCR bytes of data from Flash device into current FCM RAM buffer
		1111 RSW—Wait for $\overline{\text{LFRB}}$ to return high or time-out, then read one byte (8b port) of data from Flash device into next AS field of MDR

11.3.1.18 Flash Command Register (FCR)

The local bus Flash command register (FCR), shown in Figure 11-22, holds up to four NAND Flash EEPROM command bytes that may be referenced by opcodes in FIR during FCM operation. The values of the commands should follow the manufacturer’s datasheet for the relevant NAND Flash device.



Figure 11-22. Flash Command Register

Table 11-25 describes FCR fields.

Table 11-25. FCR Field Descriptions

Bits	Name	Description
0–7	CMD0	General purpose FCM Flash command byte 0. Opcodes in FIR that issue command index 0 write CMD0 to the NAND Flash command/data bus.
8–15	CMD1	General purpose FCM Flash command byte 1. Opcodes in FIR that issue command index 1 write CMD1 to the NAND Flash command/data bus.
16–23	CMD2	General purpose FCM Flash command byte 2. Opcodes in FIR that issue command index 2 write CMD2 to the NAND Flash command/data bus.
24–31	CMD3	General purpose FCM Flash command byte 3. Opcodes in FIR that issue command index 3 write CMD3 to the NAND Flash command/data bus.

11.3.1.19 Flash Block Address Register (FBAR)

The local bus Flash block address register (FBAR), shown in Figure 11-23, locates the NAND Flash block index for the page currently accessed.



Figure 11-23. Flash Block Address Register

Table 11-26 describes FBAR fields.

Table 11-26. FBAR Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8–31	BLK	Flash block address. The size of the NAND Flash, as configured in OR _n [PGS] and FMR[AL], determines the number of bits of BLK that are issued to the EEPROM during block address phases.

11.3.1.20 Flash Page Address Register (FPAR)

The local bus Flash page address register (FPAR), shown in Figure 11-24 and Figure 11-25, locates the current NAND Flash page in both the external NAND Flash device and FCM buffer RAM.

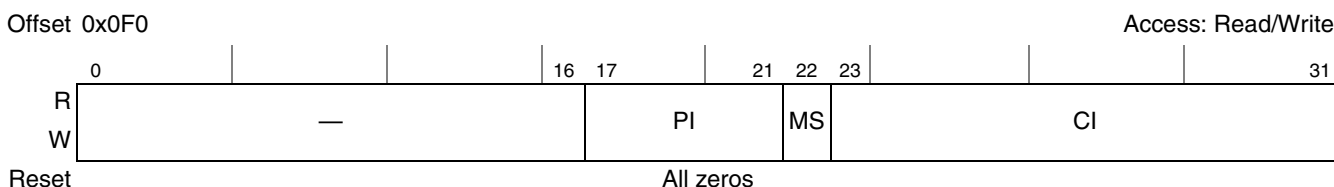


Figure 11-24. Flash Page Address Register, Small Page Device (OR_x[PGS] = 0)



Figure 11-25. Flash Page Address Register, Large Page Device (ORx[PGS] = 1)

Table 11-27 describes FPAR fields for small page devices.

Table 11-27. FPAR Field Descriptions, Small Page Device (ORx[PGS] = 0)

Bits	Name	Description
0–16	—	Reserved
17–21	PI	Page index. PI indexes the page in NAND Flash EEPROM at the current block defined by FBAR, and locates the corresponding transfer buffer in the FCM buffer RAM. The 3 LSBs of PI index one of the eight 1 Kbyte buffers in the FCM buffer RAM as follows: 000 The page is transferred to/from FCM buffer 0, address offsets 0x0000–0x03FF 001 The page is transferred to/from FCM buffer 1, address offsets 0x0400–0x07FF 010 The page is transferred to/from FCM buffer 2, address offsets 0x0800–0x0BFF 011 The page is transferred to/from FCM buffer 3, address offsets 0x0C00–0x0FFF 100 The page is transferred to/from FCM buffer 4, address offsets 0x1000–0x13FF 101 The page is transferred to/from FCM buffer 5, address offsets 0x1400–0x17FF 110 The page is transferred to/from FCM buffer 6, address offsets 0x1800–0x1BFF 111 The page is transferred to/from FCM buffer 7, address offsets 0x1C00–0x1FFF
22	MS	Main/spare region locator. In the case that FBCR[BC] = 0, MS is treated as 0. 0 Data is transferred to/from the main region of the FCM buffer; that is, the first 512 bytes of the buffer are used as the starting address. 1 Data is transferred to/from the spare region of the FCM buffer; that is, the second 512 bytes of the buffer are used as the starting address, but only an initial 16 bytes of spare region are defined.
23–31	CI	Column index. CI indexes the first byte to transfer to/from the main or spare region of the NAND Flash EEPROM and corresponding transfer buffer. In the case that FBCR[BC] = 0, CI is treated as 0. For MS = 0, CI can range 0x000–0x1FF; for MS = 1, CI can range 0x000–0x00F.

Table 11-28 describes FPAR fields for large page devices.

Table 11-28. FPAR Field Descriptions, Large Page Device (ORx[PGS] = 1)

Bits	Name	Description
0–13	—	Reserved
14–19	PI	Page index. PA indexes the page in NAND Flash EEPROM at the current block defined by FBAR, and locates the corresponding transfer buffer in the FCM buffer RAM. The LSB of PI indexes one of the two 4 Kbyte buffers in the FCM buffer RAM as follows: 0 The page is transferred to/from FCM buffer 0, address offsets 0x0000–0x0FFF 1 The page is transferred to/from FCM buffer 1, address offsets 0x1000–0x1FFF

Table 11-28. FPAR Field Descriptions, Large Page Device (ORx[PGS] = 1) (continued)

Bits	Name	Description
20	MS	Main/spare region locator. In the case that FBCR[BC] = 0, MS is treated as 0. 0 Data is transferred to/from the main region of the FCM buffer; that is, the first 2048 bytes of the buffer are used as the starting address. 1 Data is transferred to/from the spare region of the FCM buffer; that is, the second 2048 bytes of the buffer are used as the starting address, but only an initial 64 bytes of spare region are defined.
21–31	CI	Column index. CI indexes the first byte to transfer to/from the main or spare region of the NAND Flash EEPROM and corresponding transfer buffer. In the case that FBCR[BC] = 0, CI is treated as 0. For MS = 0, CI can range 0x000–0x7FF; for MS = 1, CI can range 0x000–0x03F.

11.3.1.21 Flash Byte Count Register (FBCR)

The local bus Flash byte count register (FBCR), shown in [Figure 11-26](#), defines the size of FCM block transfers for reads and writes to the NAND Flash EEPROM.

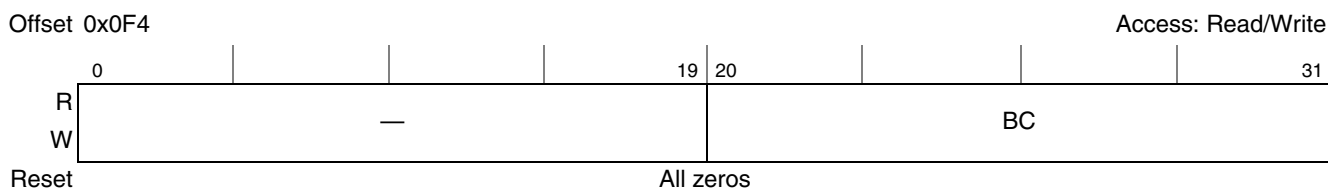


Figure 11-26. Flash Byte Count Register

[Table 11-29](#) describes FBCR fields.

Table 11-29. FBCR Field Descriptions

Bits	Name	Description
0–19	—	Reserved
20–31	BC	Byte count determines how many bytes are transferred by the FCM during data read (RB) or data write (WB) opcodes. The first byte accessed in the NAND Flash EEPROM is located by the FPAR register, and successive bytes are transferred until either BC bytes have been counted, or the end of the spare region of the currently addressed Flash page has been reached. If BC = 0, an entire Flash page and its spare region will be transferred by FCM, in which case FPAR[MS] and FPAR[CI] are treated as zero regardless of their values. BC = 0 is the only setting that permits FCM to generate and check ECC.

11.4 Functional Description

The eLBC allows the implementation of memory systems with very specific timing requirements.

- The GPCM provides interfacing for simpler, lower-performance memories and memory-mapped devices. It has inherently lower performance because it does not support bursting. For this reason, GPCM-controlled banks are used primarily for boot-loading from NVRAM or NOR Flash, and access to low-performance memory-mapped peripherals.
- The FCM interfaces the eLBC to NAND Flash EEPROMs with 8-bit data bus. The FCM has an automatic boot-loading feature that allows the CPU to boot from high density EEPROM, loading

the boot block into 4 Kbytes of RAM for execution of the first level boot code. Following boot, FCM provides a flexible instruction sequencer that allows a user-defined command, address, and data transfer sequence of up to 8 steps to be executed against a memory-mapped buffer RAM. Programmable set-up time, hold time, and wait states permit the FCM to maximize the performance of NAND Flash block transfers, which can proceed in parallel with software processing of the multiple RAM buffers. A single-pass ECC engine in the FCM permits zero-overhead error checking, reporting, and correction in both boot blocks and page data transfers if enabled.

- The UPM supports refresh timers, address multiplexing of the external bus, and generation of programmable control signals for row address and column address strobes, to allow for a minimal glue logic interface to DRAMs, burstable SRAMs, and almost any other kind of peripheral with asynchronous timing or single data rate clocking. The UPM can be used to generate flexible, user-defined timing patterns for control signals that govern a memory device. These patterns define how the external control signals behave during a read, write, burst-read, or burst-write access. Refresh timers are also available to periodically initiate user-defined refresh patterns.

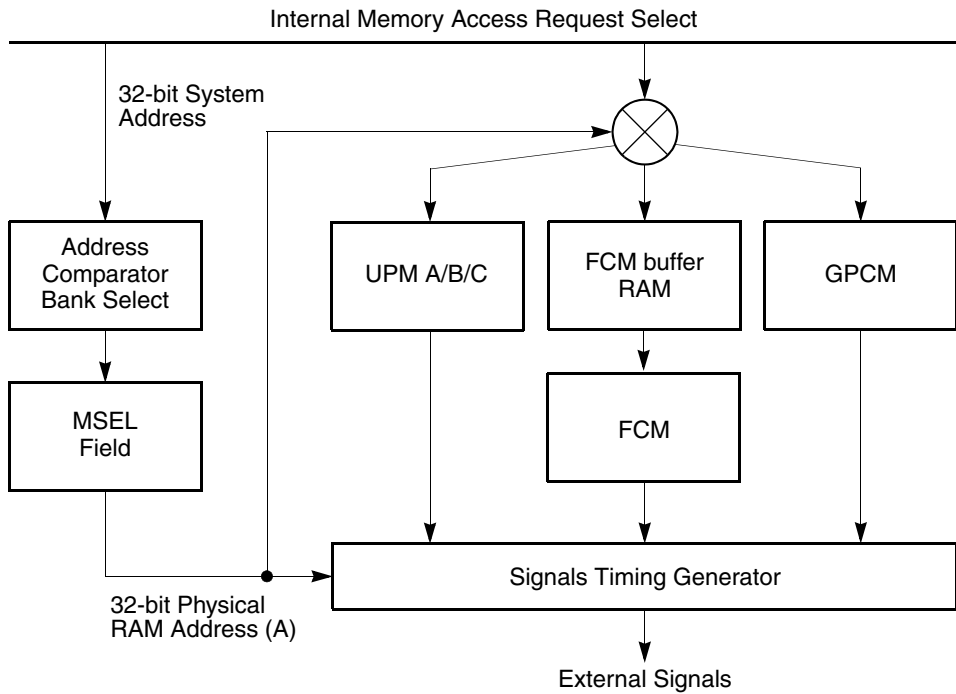


Figure 11-27. Basic Operation of Memory Controllers in the eLBC

Each memory bank (chip select) can be assigned to any of these three types of machines through the machine select bits of the base register for that bank ($BR_n[MSEL]$), as illustrated in [Figure 11-27](#). If a bank match occurs, the corresponding machine (GPCM, FCM, or UPM) then takes ownership of the external signals that control the access and maintains control until the transaction ends.

NOTE

Different machines (FCM/GPCM/UPM) share the address, data, and control signals. While the eLBC is servicing a transaction, subsequent transactions are queued until the current transaction has completed.

11.4.1 Basic Architecture

The following subsections describe the basic architecture of the eLBC.

11.4.1.1 Address and Address Space Checking

The defined base addresses are written to the BR_n registers, while the corresponding address masks are written to the OR_n registers. Each time a local bus access is requested, the internal transaction address is compared with each bank. Addresses are decoded by comparing the 17 MSBs of the address, masked by $OR_n[AM]$, with the base address for each bank ($BR_n[BA]$). If a match is found on a memory controller bank, the attributes defined in the BR_n and OR_n for that bank are used to control the memory access. If a match is found in more than one bank, the lowest-numbered bank handles the memory access (that is, bank 0 has priority over bank 1).

11.4.1.2 External Address Latch Enable Signal (LALE)

The local bus uses a multiplexed address/data bus. Therefore the eLBC must distinguish between address and data phases, which take place on the same bus (LAD pins). The LALE signal, when asserted, signifies an address phase during which the eLBC drives the memory address on the LAD pins. An external address latch uses this signal to capture the address and provide it to the address pins of the memory or peripheral device. When LALE is negated, LAD then serves as the (bi-directional) data bus for the access. Any address phase initiates the assertion of LALE, which has a programmable duration of between 1 and 4 bus clock cycles.

To ensure adequate hold time on the external address latch, LALE negates earlier than the address changes on LAD during address phases. By default, LALE negates earlier by 1 platform clock period. For example, if the platform clock is operating at 133 MHz, then 7.5 ns of address hold time is introduced. However, at higher frequencies, the duration of the shortened LALE pulse may not meet the minimum latch enable pulse width specifications of some latches. In such cases, setting $LBCR[AHD] = 1$ increases the LALE pulse width by 1 platform clock cycle, but decreases the address hold time by the same amount. If both longer hold time and longer LALE pulse duration are needed, then the address phase can be extended using the $OR_n[EAD]$ and $LCRR[EADC]$ fields, and the $LBCR[AHD]$ bit can be left at 0. However, this will add latency to all address tenures.

The frequency of LALE assertion varies across the three memory controllers:

- For GPCM, every assertion of $\overline{LCS_n}$ 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 $\overline{LCS_n}$ 32 times in order to satisfy a 32-byte cache line transfer.
- For FCM, LALE asserts prior to each multi-command operation sequence, but LALE can be ignored on NAND Flash EEPROM accesses as the signal does *not* enable address latching in such

devices. The value on the LAD and LA pins during LALE assertion is driven low-impedance, but otherwise not defined for FCM banks.

- In the case of UPM, the frequency of LALE assertion depends on how the UPM RAM is programmed. UPM single accesses typically assert LALE once, upon commencement, but it is possible to program UPM to assert LALE several times, and to change the values of LA_n with and without LALE being involved.

In general, when using the GPCM controller it is not necessary to use LA if a sufficiently wide latch is used to capture the entire address during LALE phases. The UPMs may require LA if the eLBC is generating its own burst address sequence.

To illustrate how a large transaction is handled by the eLBC, [Figure 11-28](#) shows eLBC signals for the GPCM performing a 32-byte write starting at address 0x5420.

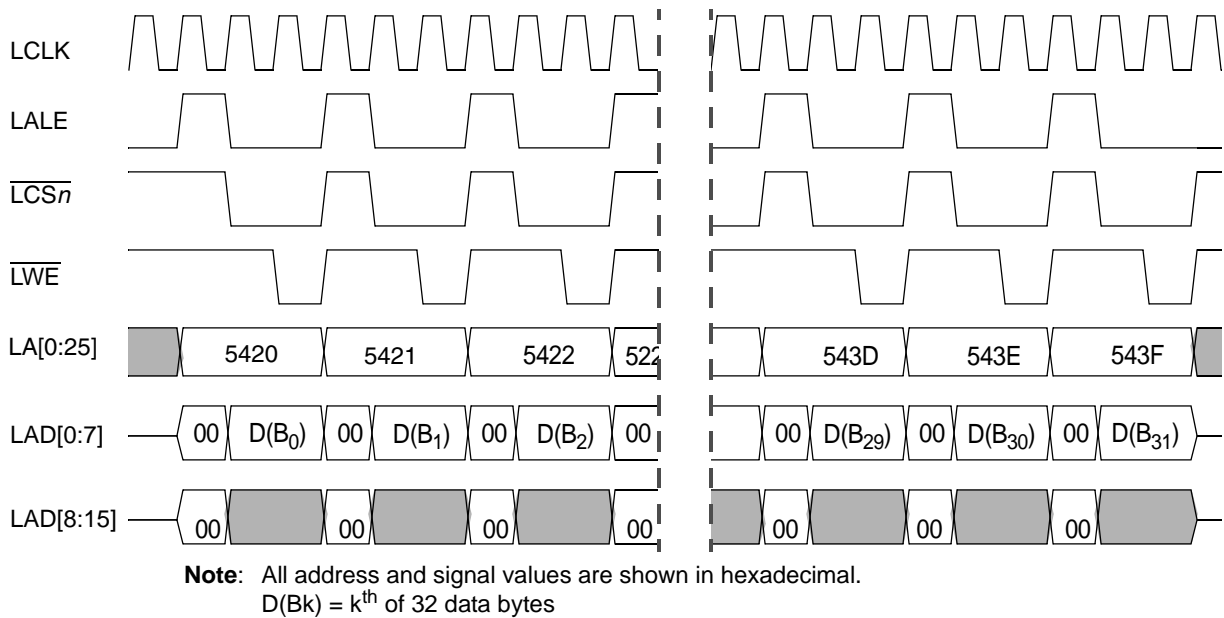


Figure 11-28. Example of 8-Bit GPCM Writing 32 Bytes to Address 0x5420 (LCRR[PBYFP] = 0)

If the RCW is loaded by the eLBC, LALE may remain at an unknown value for up to 8 cycles after PORESET negation. In general, it is recommended that a latch be implemented for this adjustment and not a state machine triggered by LALE.

If the RCW is not loaded by the eLBC (for example, I2C or hard-coded options are used), then LALE is at an unknown value until the PLL is locked and should be ignored until the negation of HRESET.

11.4.1.3 Data Transfer Acknowledge (TA)

The three memory controllers in the eLBC generate an internal transfer acknowledge signal, TA, to allow data on LAD to be either sampled (for reads) or changed (on writes). The data sampling/data change always occurs at the end of the bus cycle in which the eLBC asserts TA internally. In eLBC debug mode, TA is also visible externally on the LDVAL pin. The GPCM controller automatically generates TA according to the timing parameters programmed for them in the option and mode registers; FCM generates

TA whenever data read and write instructions are executed out of register FIR; a UPM generates TA only when a UPM pattern has the UTA RAM word bit set. Figure 11-29 shows LALE, TA (internal), and \overline{LCSn} . Note that TA and LALE are never asserted together, and that for the duration of LALE, \overline{LCSn} (or any other control signal) remains negated or frozen.

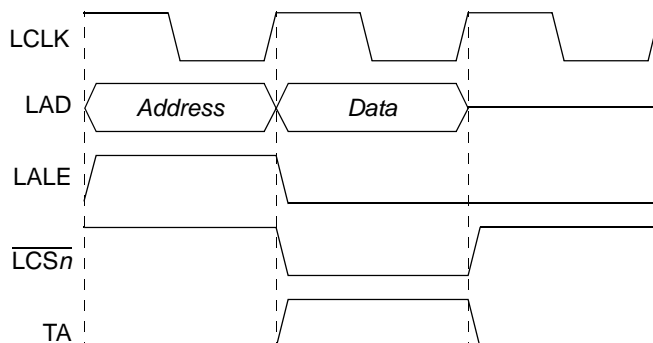


Figure 11-29. Basic eLBC Bus Cycle with LALE, TA, and \overline{LCSn}

11.4.1.4 Data Buffer Control (LBCTL)

The memory controller provides a data buffer control signal for the local bus (LBCTL). This signal is activated when a GPCM-, FCM-, or UPM-controlled bank is accessed. LBCTL can be disabled by setting $ORn[BCTLD]$. LBCTL can be further configured by $LBCR[BCTLTC]$ to act as an extra \overline{LWE} or an extra \overline{LOE} signal when in GPCM mode.

If LBCTL is configured as a data buffer control ($LBCR[BCTLTC] = 00$), 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.

11.4.1.5 Bus Monitor

A bus monitor is provided to ensure that each transaction is terminated within a reasonable (user defined) period. When a transaction starts, the bus monitor starts counting down from the time-out value ($LBCR[BMT] \times LBCR[BMTPS]$) until a data beat is acknowledged on the bus. It then reloads the time-out value and resumes the countdown until the data tenure completes and then idles if there is no pending transaction. Setting $LTEDR[BMD]$ disables bus monitor error checking (that is, the $LTESR[BM]$ bit is not set by a bus monitor time-out); however, the bus monitor is still active and can generate a UPM exception (as noted in Section 11.4.4.1.4, “Exception Requests,”) or terminate a GPCM access.

It is very important to ensure that the value of $LBCR[BMT]$ is not set too low; otherwise spurious bus time-outs may occur during normal operation—resulting in incomplete data transfers. Accordingly, the time-out value represented by the $LBCR[BMT]$, $LBCR[BMTPS]$ pair must not be set below 40 bus cycles for time-out under any circumstances.

NOTE

When the FCM is in the middle of a long transaction (such as NAND erase, write, etc.), another transaction on the GPCM or UPM will trigger the bus monitor to start, even though the GPCM or UPM is waiting for the FCM to finish. If the bus monitor times out, it could corrupt the current NAND flash operation as well as terminate the GPCM or UPM operation. To avoid such cases, it is recommended that while using FCM the bus monitor timeout be programmed to its maximum setting of $LBCR[BMT] = 0$ and $LBCR[BMTPS] = 0xF$.

11.4.2 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— BRn and ORn .

Figure 11-30 shows a simple connection between an 8-bit port size SRAM device and the eLBC in GPCM mode. Byte-write enable signals (\overline{LWE}) are available for each byte written to memory. Also, the output enable signal (\overline{LOE}) is provided to minimize external glue logic. On system reset, a global (boot) chip-select is available that provides a boot ROM chip-select ($\overline{LCS0}$) prior to the system being fully configured.

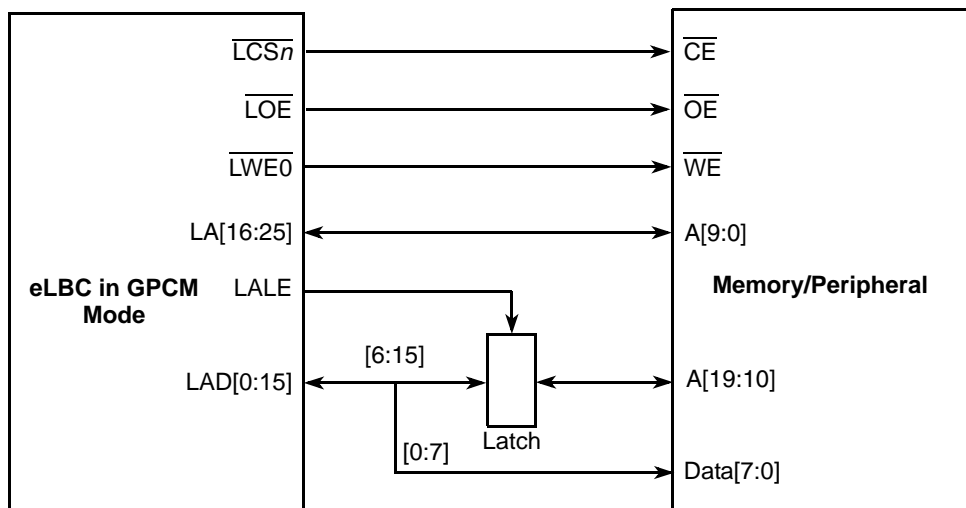


Figure 11-30. Enhanced Local Bus to GPCM Device Interface

Figure 11-31 shows \overline{LCS} as defined by the setup time required between the address lines and \overline{CE} . The user can configure $ORn[ACS]$ to specify \overline{LCS} to meet this requirement. Generally, the attributes for the

memory cycle are taken from OR_n . These attributes include the CSNT, ACS, XACS, SCY, TRLX, EHTR, and SETA fields.

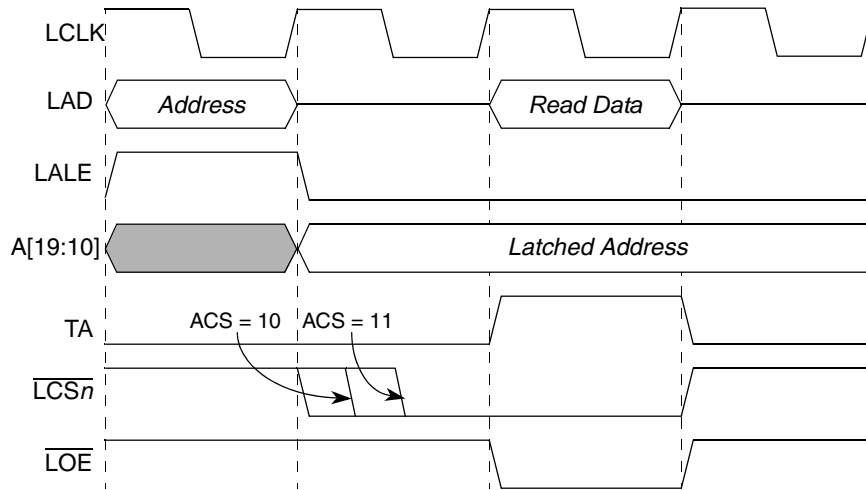


Figure 11-31. GPCM Basic Read Timing (XACS = 0, ACS = 1x, TRLX = 0, CLKDIV = 4,8)

11.4.2.1 GPCM Read Signal Timing

The basic GPCM read timing parameters that may be set by the OR_n attributes are shown in Figure 11-32. The read access cycle commences upon latching of the memory address (LALE negated), and concludes when LBCTL returns high to turn the local bus around for a subsequent address phase. Read data is captured by eLBC on the falling edge of TA. \overline{LOE} and \overline{LCSn} negate high simultaneously, in some cases before the end of the read access to provide additional hold time for the external memory.

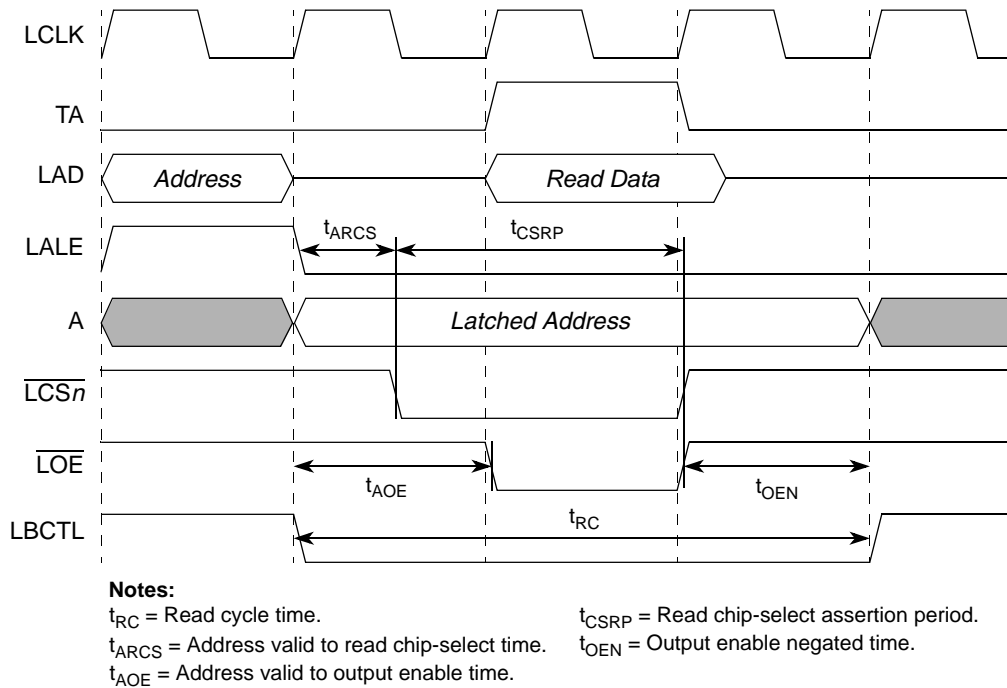


Figure 11-32. GPCM General Read Timing Parameters

Table 11-30 lists the signal timing parameters for a GPCM read access as the option register attributes are varied.

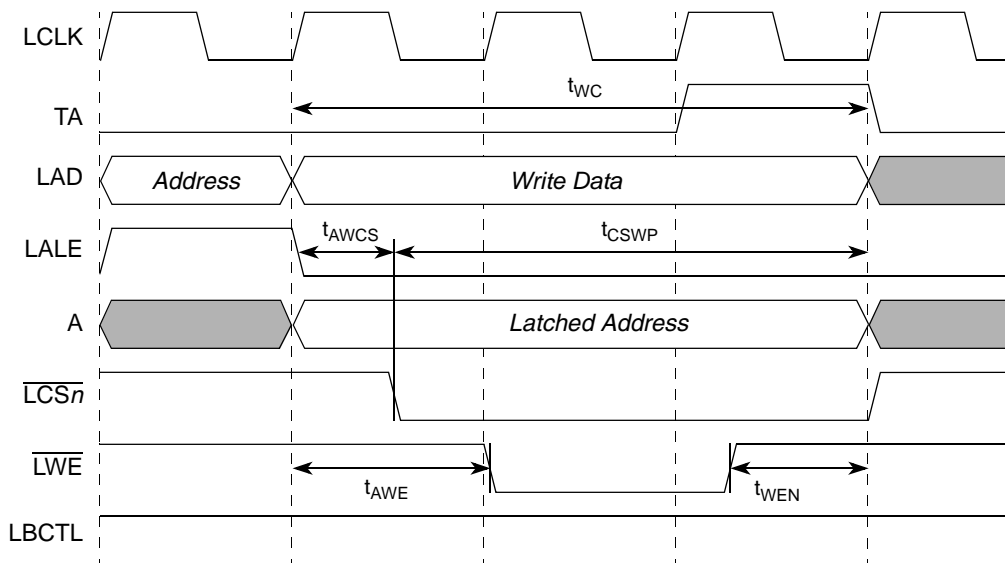
Table 11-30. GPCM Read Control Signal Timing

Option Register Attributes				Signal Timing (LCLK clock cycles) ¹				
TRLX	EHTR	XACS	ACS	t _{ARCS}	t _{CSRP}	t _{AOE}	t _{OEN}	t _{RC}
0	0	0	0X	0	2 + SCY	1	0	2 + SCY
0	0	0	10	¼ (½)	1¾ + SCY (2+SCY)	1	0	2 + SCY
0	0	0	11	½	1½ + SCY	1	0	2 + SCY
0	0	1	0X	0	2 + SCY	1	0	2 + SCY
0	0	1	10	1	1 + SCY	1	0	2 + SCY
0	0	1	11	2	1 + SCY	2	0	3 + SCY
0	1	0	0X	0	2 + SCY	1	1	3 + SCY
0	1	0	10	¼ (½)	1¾ + SCY (1½+SCY)	1	1	3 + SCY
0	1	0	11	½	1½ + SCY	1	1	3 + SCY
0	1	1	0X	0	2 + SCY	1	1	3 + SCY
0	1	1	10	1	1 + SCY	1	1	3 + SCY
0	1	1	11	2	1 + SCY	2	1	4 + SCY
1	0	0	0X	0	2 + 2 × SCY	1	4	6 + 2 × SCY
1	0	0	10	1¼ (1½)	1¾ + 2 × SCY (1½+2×SCY)	2	4	7 + 2 × SCY
1	0	0	11	1½	1½ + 2 × SCY	2	4	7 + 2 × SCY
1	0	1	0X	0	2 + 2 × SCY	1	4	6 + 2 × SCY
1	0	1	10	2	1 + 2 × SCY	2	4	7 + 2 × SCY
1	0	1	11	3	1 + 2 × SCY	3	4	8 + 2 × SCY
1	1	0	0X	0	2 + 2 × SCY	1	8	10 + 2 × SCY
1	1	0	10	1¼ (1½)	1¾ + 2 × SCY (1½+2×SCY)	2	8	11 + 2 × SCY
1	1	0	11	1½	1½ + 2 × SCY	2	8	11 + 2 × SCY
1	1	1	0X	0	2 + 2 × SCY	1	8	10 + 2 × SCY
1	1	1	10	2	1 + 2 × SCY	2	8	11 + 2 × SCY
1	1	1	11	3	1 + 2 × SCY	3	8	12 + 2 × SCY

¹ Times in parentheses are specific for the case LCRR[CLKDIV] = 2; other times apply to all CLKDIV values.

11.4.2.2 GPCM Write Signal Timing

The basic GPCM write timing parameters that may be set by the OR_n attributes are shown in Figure 11-33. The write access cycle commences upon latching of the memory address (LALE negated), and concludes when \overline{LCSn} returns high. LBCTL remains stable for the entire cycle to drive data onto any secondary data bus. Write data becomes invalid following the falling edge of TA. \overline{LWE} may, in some cases, negate high before the end of the write access to provide additional hold time for the external memory.



Notes:
 t_{WC} = Write cycle time. t_{CSWP} = Write chip-select assertion period.
 t_{AWCS} = Address valid to write chip-select time. t_{WEN} = Write enable negated time wrt chip-sele.
 t_{AWE} = Address valid to write enable time.

Figure 11-33. GPCM General Write Timing Parameters

Table 11-31 lists the signal timing parameters for a GPCM write access as the option register attributes are varied.

Table 11-31. GPCM Write Control Signal Timing

Option Register Attributes				Signal Timing (LCLK clock cycles) ¹				
TRLX	XACS	ACS	CSNT	t_{AWCS}	t_{CSWP}	t_{AWE}	t_{WEN}	t_{WC}
0	0	00	0	0	2 + SCY	1	0	2 + SCY
0	0	10	0	$\frac{1}{4}$ ($\frac{1}{2}$)	$1\frac{3}{4}$ + SCY (2+SCY)	1	0	2 + SCY
0	0	11	0	$\frac{1}{2}$	$1\frac{1}{2}$ + SCY	1	0	2 + SCY
0	1	00	0	0	2 + SCY	1	0	2 + SCY
0	1	10	0	1	1 + SCY	1	0	2 + SCY
0	1	11	0	2	1 + SCY	2	0	3 + SCY
0	0	00	1	0	2 + SCY	1	$\frac{1}{4}$ (0)	2 + SCY

Table 11-31. GPCM Write Control Signal Timing (continued)

Option Register Attributes				Signal Timing (LCLK clock cycles) ¹				
TRLX	XACS	ACS	CSNT	t _{AWCS}	t _{CSWP}	t _{AWE}	t _{WEN}	t _{wc}
0	0	10	1	¼ (½)	1½ + SCY	1	0	1¾ + SCY (1½+SCY)
0	0	11	1	½	1¼ + SCY (1+SCY)	1	0	1¾ + SCY (1½+SCY)
0	1	00	1	0	2 + SCY	1	¼ (0)	2 + SCY
0	1	10	1	1	¾ + SCY (½+SCY)	1	0	1¾ + SCY (1½+SCY)
0	1	11	1	2	¾ + SCY (½+SCY)	2	0	2¾ + SCY (2½+SCY)
1	0	00	0	0	2 + 2 × SCY	1	0	2 + 2 × SCY
1	0	10	0	1¼ (1½)	1¾ + 2 × SCY (2+2×SCY)	2	0	3 + 2 × SCY
1	0	11	0	1½	1½ + 2 × SCY	2	0	3 + 2 × SCY
1	1	00	0	0	2 + 2 × SCY	1	0	2 + 2 × SCY
1	1	10	0	2	1 + 2 × SCY	2	0	3 + 2 × SCY
1	1	11	0	3	1 + 2 × SCY	3	0	4 + 2 × SCY
1	0	00	1	0	3 + 2 × SCY	1	1¼ (1)	3 + 2 × SCY
1	0	10	1	1¼ (1½)	1½ + 2 × SCY	2	0	2¾ + 2 × SCY (2½+2×SCY)
1	0	11	1	1½	1¼ + 2 × SCY (1+2×SCY)	2	0	2¾ + 2 × SCY (2½+2×SCY)
1	1	00	1	0	3 + 2 × SCY	1	1¼ (1)	3 + 2 × SCY
1	1	10	1	2	¾ + 2 × SCY (½+2×SCY)	2	0	2¾ + 2 × SCY (2½+2×SCY)
1	1	11	1	3	¾ + 2 × SCY (½+2×SCY)	3	0	3¾ + 2 × SCY (3½+2×SCY)

¹ Times in parentheses are specific for the case LCRR[CLKDIV] = 2; other times apply to all CLKDIV values.

11.4.2.3 Chip-Select Assertion Timing

The banks selected to work with the GPCM support an option to drive the \overline{LCSn} signal with different timings (with respect to the external address/data bus). \overline{LCSn} can be driven in any of the following ways:

- Simultaneous with the latched memory address. (This refers to the externally latched address and not the address timing on LAD. That is, the chip select does not assert during LALE).
- One quarter of a clock cycle later (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.

The timing diagram in [Figure 11-31](#) shows two chip-select assertion timings for the case LCRR[CLKDIV] = 4 or 8. If LCRR[CLKDIV] = 2, \overline{LCSn} asserts identically for OR_n[ACS] = 10 or 11.

11.4.2.3.1 Programmable Wait State Configuration

The GPCM supports internal generation of transfer acknowledge. It allows between zero and 30 wait states to be added to an access by programming OR_n[SCY] and OR_n[TRLX]. Internal generation of transfer acknowledge is enabled if OR_n[SETA] = 0. If \overline{LGTA} is asserted externally two bus clock cycles or more before the wait state counter has expired (to allow for synchronization latency), the current memory cycle is terminated by \overline{LGTA} ; otherwise it is terminated by the expiration of the wait state counter. Regardless of the setting of OR_n[SETA], wait states prolong the assertion duration of both \overline{LOE} and \overline{LWEn} in the same manner. When TRLX = 1, the number of wait states inserted by the memory controller is doubled from OR_n[SCY] cycles to 2 × OR_n[SCY] cycles, allowing a maximum of 30 wait states.

11.4.2.3.2 Chip-Select and Write Enable Negation Timing

[Figure 11-30](#) shows a basic connection between the local bus and a static memory device. In this case, \overline{LCSn} is connected directly to \overline{CE} of the memory device. The $\overline{LWE}[0:1]$ signals are connected to the respective $\overline{WE}[1:0]$ signals on the memory device where each $\overline{LWE}[0:1]$ signal corresponds to a different data byte.

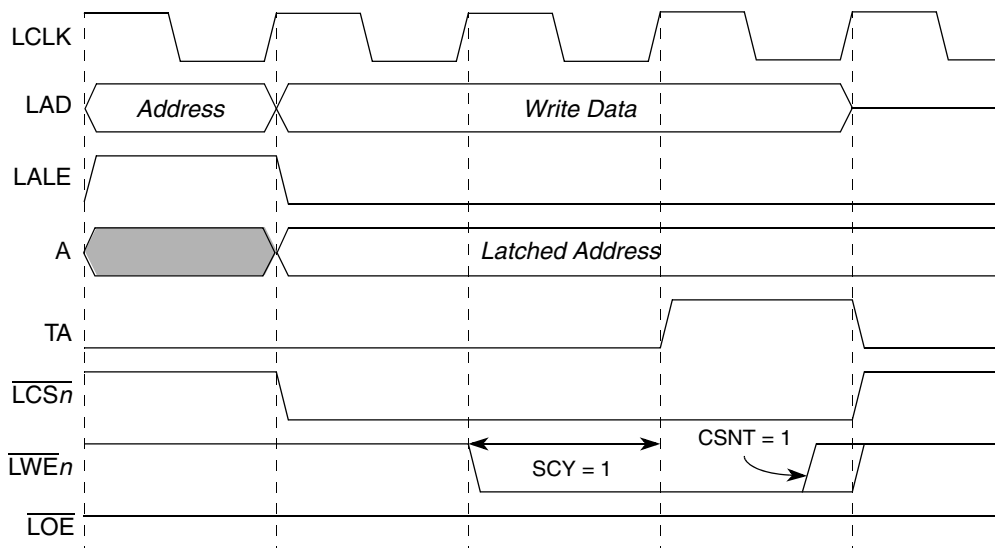


Figure 11-34. GPCM Basic Write Timing
(XACS = 0, ACS = 00, CSNT = 1, SCY = 1, TRLX = 0, CLKDIV = 4, 8)

As [Figure 11-34](#) shows, the timing for \overline{LCSn} is the same as for the latched address. The strobes for the transaction are supplied by \overline{LOE} or \overline{LWEn} , depending on the transaction direction—read or write (write

case shown in the figure). $OR_n[CSNT]$, along with $OR_n[TRLX]$, control the timing for the appropriate strobe negation in write cycles. When this attribute is asserted, the strobe is negated one quarter of a clock before the normal case provided that $LCRR[CLDIV] = 4$ or 8 . For example, when $ACS = 00$ and $CSNT = 1$, \overline{LWEn} is negated one quarter of a clock earlier, as shown in Figure 11-34. If $LCRR[CLDIV] = 2$, \overline{LWEn} is negated either coincident with \overline{LCSn} or one cycle earlier.

1. \overline{LCSn} is affected by $CSNT$ and $TRLX$ only if $ACS[0]$ is non zero. However, \overline{LWEn} is affected independent of ACS .
2. When $CSNT$ attribute is asserted, the strobe is negated one quarter of a clock before the normal case provided that $LCRR[CLDIV] = 4$ or 8 .
3. $TRLX = 1$ in conjunction with $CSNT = 1$, negates the \overline{LCSn} and \overline{LWEn} $1 + 1/4$ cycle earlier if $LCRR[CLKDIV] = 4$ or 8 .

If $LCRR[CLKDIV] = 2$, \overline{LCSn} and \overline{LWEn} are negated either normally or one cycle earlier if $TRLX = 1$.

For example, when $ACS = 00$, $CSNT = 1$ and $TRLX = 0$, \overline{LWEn} is negated one quarter of a clock earlier and \overline{LCSn} is negated normally as shown in Figure 11-34.

11.4.2.3.3 Relaxed Timing

$OR_x[TRLX]$ is provided for memory systems that require more relaxed timing between signals. Setting $TRLX = 1$ has the following effect on timing:

- An additional bus cycle is added between the address and control signals (but only if ACS is not equal to 00).
- The number of wait states specified by SCY is doubled, providing up to 30 wait states.
- The extended hold time on read accesses ($EHTR$) is extended further.
- \overline{LCSn} signals are negated one cycle earlier during writes (but only if ACS is not equal to 00).
- $\overline{LWE}[0:1]$ signals are negated one cycle earlier during writes.

Figure 11-35 and Figure 11-36 show relaxed timing read and write transactions. The effect of $LCRR[CLKDIV] = 2$ for these examples is only to delay the assertion of \overline{LCSn} in the $ACS = 10$ case to

the ACS = 11 case. The example in Figure 11-36 also shows address and data multiplexing on LAD for a pair of writes issued consecutively.

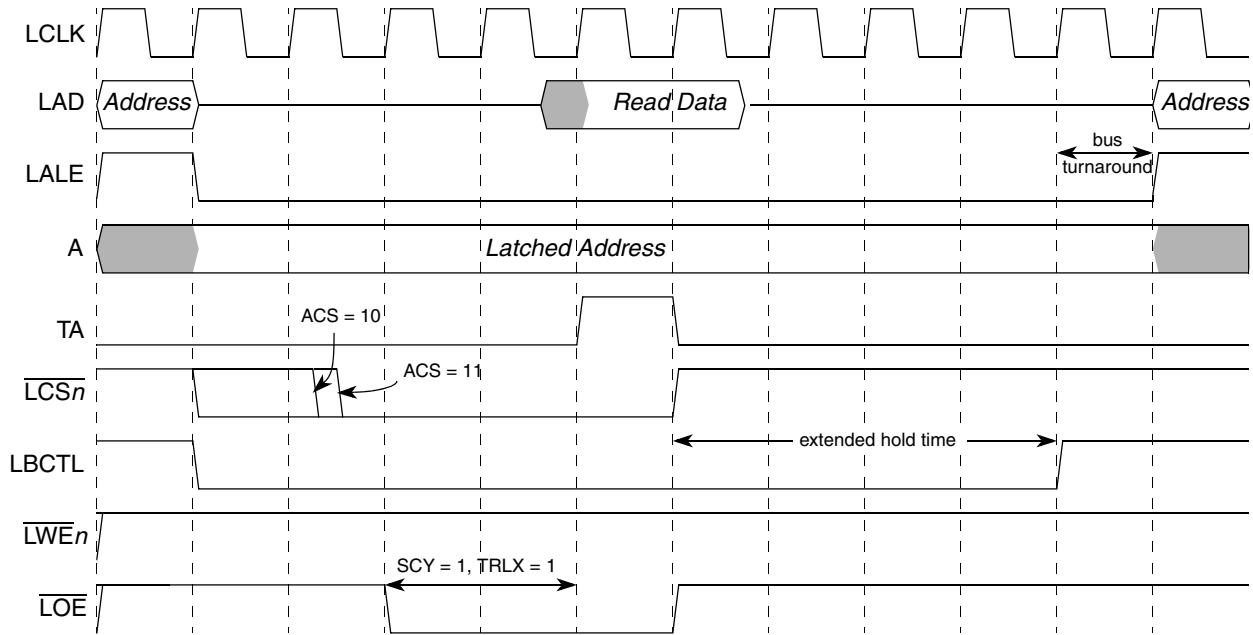


Figure 11-35. GPCM Relaxed Timing Back-to-Back Reads
 (XACS = 0, ACS = 1x, SCY = 1, CSNT = 0, TRLX = 1, EHTR = 0, CLKDIV = 4, 8)

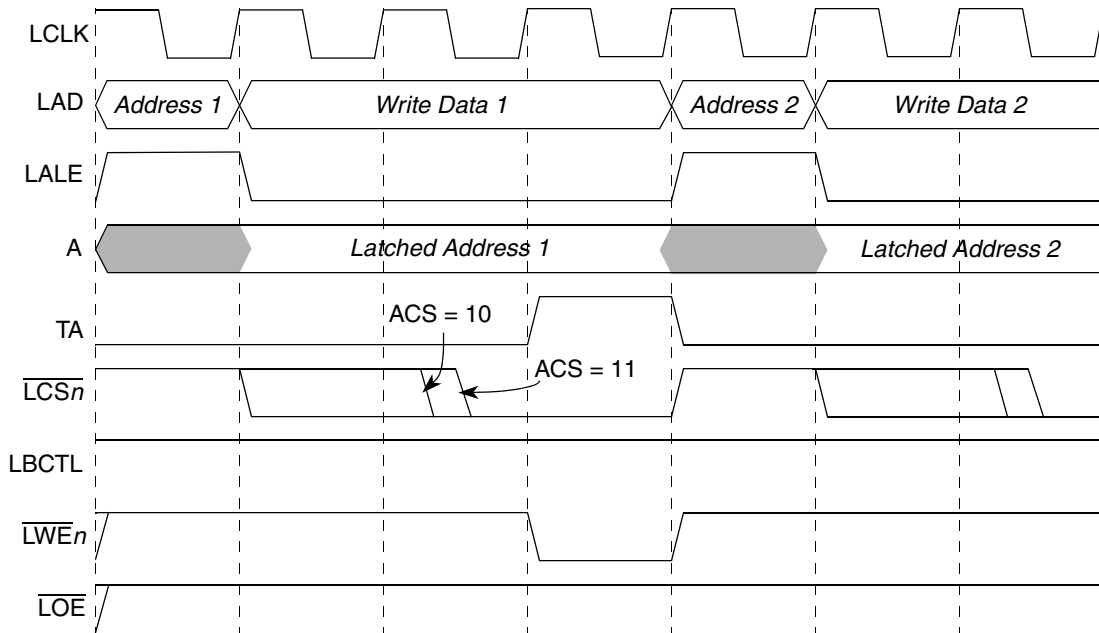


Figure 11-36. GPCM Relaxed Timing Back-to-Back Writes
 (XACS = 0, ACS = 1x, SCY = 0, CSNT = 0, TRLX = 1, CLKDIV = 4, 8)

When TRLX and CSNT are set in a write access, the $\overline{LWE}[0:1]$ strobe signals are negated one clock earlier than in the normal case, as shown in Figure 11-37 and Figure 11-38. If $ACS \neq 00$, \overline{LCSn} is also negated one clock earlier.

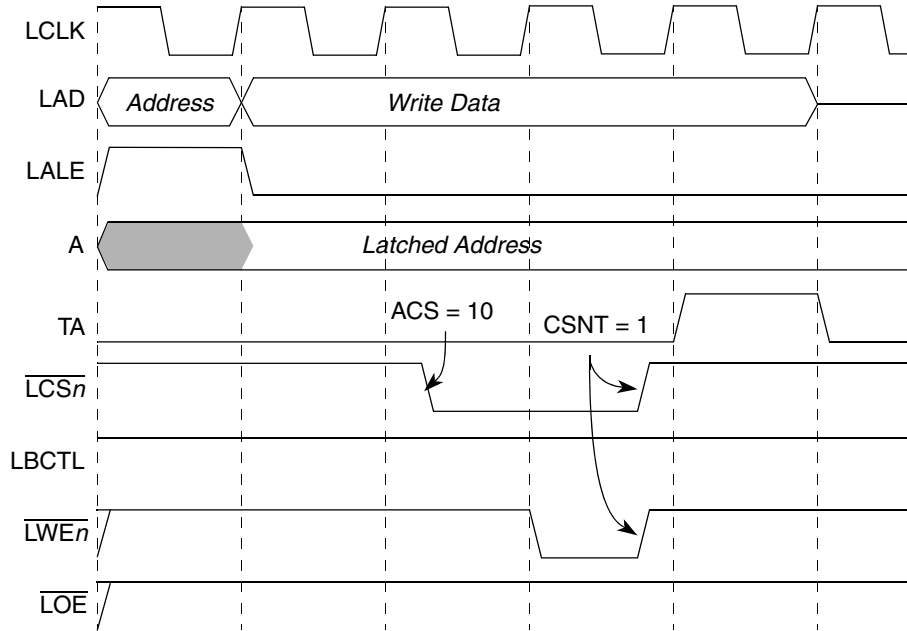


Figure 11-37. GPCM Relaxed Timing Write
(XACS = 0, ACS = 10, SCY = 0, CSNT = 1, TRLX = 1, CLKDIV = 4, 8)

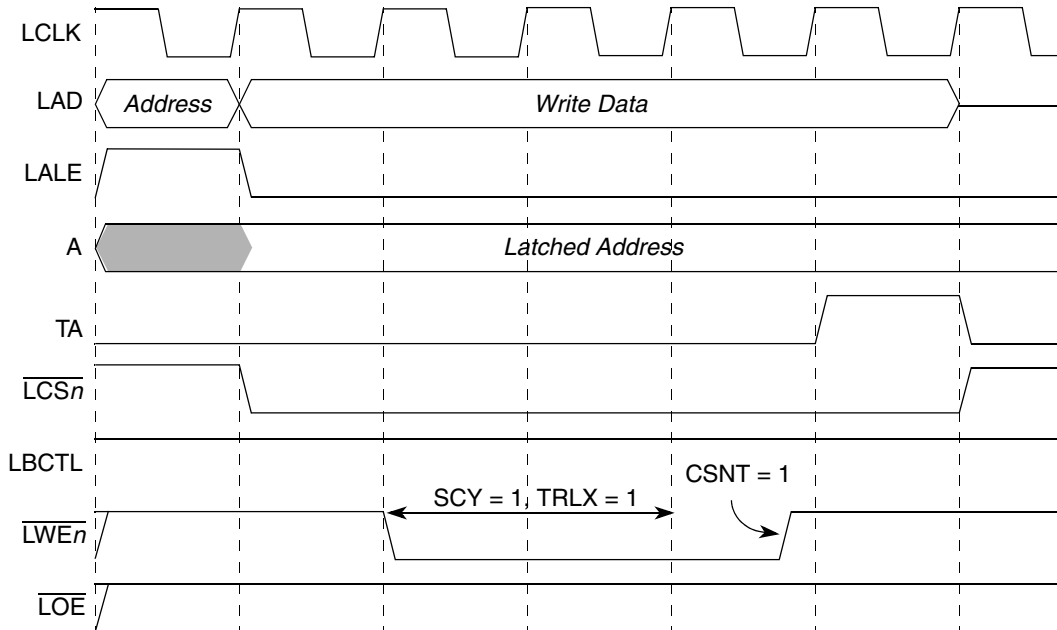


Figure 11-38. GPCM Relaxed Timing Write
(XACS = 0, ACS = 00, SCY = 1, CSNT = 1, TRLX = 1, CLKDIV = 4, 8)

11.4.2.3.4 Output Enable (\overline{LOE}) Timing

The timing of the \overline{LOE} is affected only by $TRLX$. It always asserts and negates on the rising edge of the bus clock. \overline{LOE} asserts either on the rising edge of the bus clock after \overline{LCSn} is asserted or coinciding with \overline{LCSn} (if $XACS = 1$ and $ACS = 10$ or $ACS = 11$). Accordingly, assertion of \overline{LOE} can be delayed (along with the assertion of \overline{LCSn}) by programming $TRLX = 1$. \overline{LOE} negates on the rising clock edge coinciding with \overline{LCSn} negation

11.4.2.3.5 Extended Hold Time on Read Accesses

Slow memory devices that take a long time to disable their data bus drivers on read accesses should choose some combination of $ORn[TRLX,EHTR]$. Any access following a read access to the slower memory bank is delayed by the number of clock cycles specified in [Table 11-7](#) in addition to any existing bus turnaround cycle. The final bus turnaround cycle is automatically inserted by the eLBC for reads, regardless of the setting of $ORn[EHTR]$.

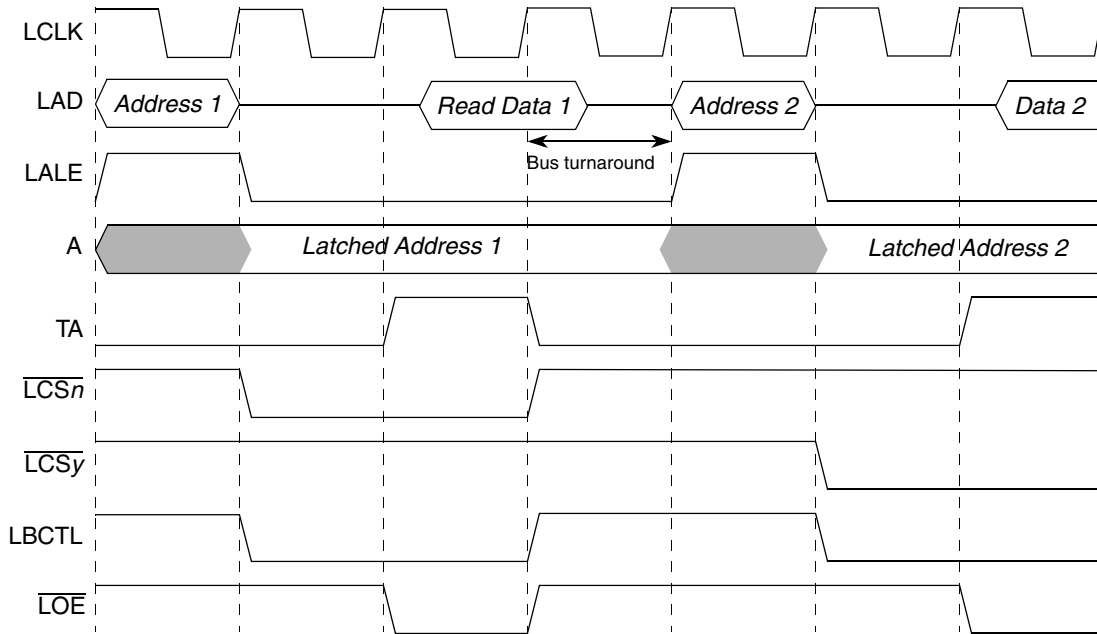


Figure 11-39. GPCM Read Followed by Read ($TRLX = 0$, $EHTR = 0$, Fastest Timing)

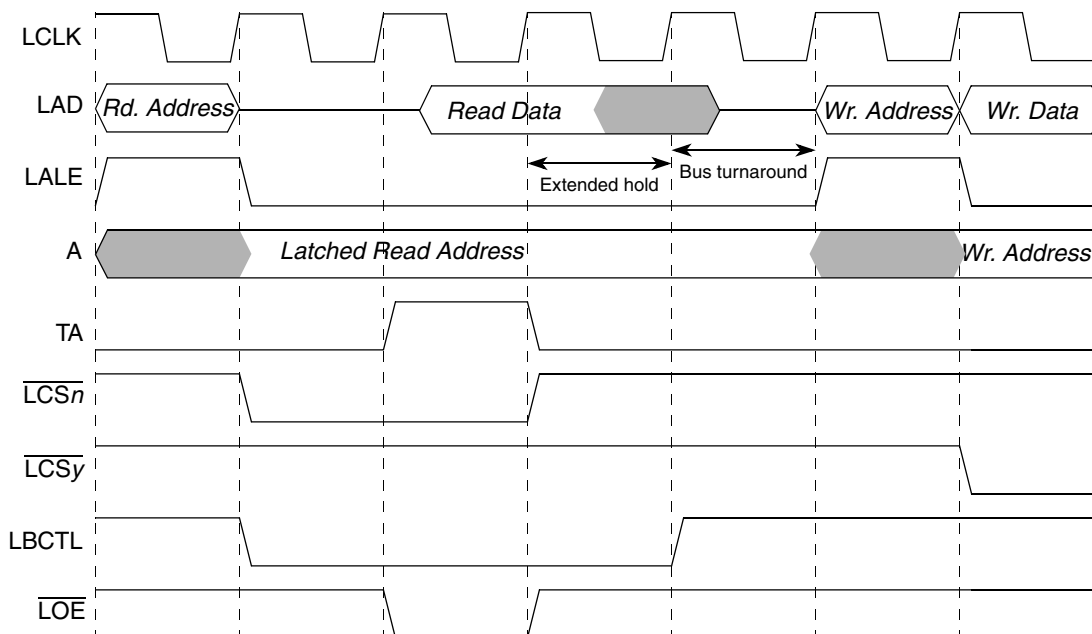


Figure 11-40. GPCM Read Followed by Write
(TRLX = 0, EHTR = 1, One-Cycle Extended Hold Time on Reads)

11.4.2.4 External Access Termination ($\overline{\text{LGTA}}$)

External access termination is supported by the GPCM using the asynchronous $\overline{\text{LGTA}}$ input signal, which is synchronized and sampled internally by the local bus. If, during assertion of $\overline{\text{LCSn}}$, the sampled $\overline{\text{LGTA}}$ signal is asserted, it is converted to an internal generation of transfer acknowledge, which terminates the current GPCM access (regardless of the setting of $\text{ORn}[\text{SETA}]$). $\overline{\text{LGTA}}$ should be asserted for at least one bus cycle to be effective. Note that because $\overline{\text{LGTA}}$ is synchronized, bus termination occurs two cycles after $\overline{\text{LGTA}}$ assertion, so in case of read cycle, the device still must drive data as long as $\overline{\text{LOE}}$ is asserted.

The user selects whether transfer acknowledge is generated internally or externally ($\overline{\text{LGTA}}$) by programming $\text{ORn}[\text{SETA}]$. Asserting $\overline{\text{LGTA}}$ always terminates an access, even if $\text{ORn}[\text{SETA}] = 0$.

(internal transfer acknowledge generation), but it is the only means by which an access can be terminated if $OR_n[SETA] = 1$.

In PLL bypass mode, the timing of \overline{LGTA} is illustrated by the example in [Figure 11-41](#).

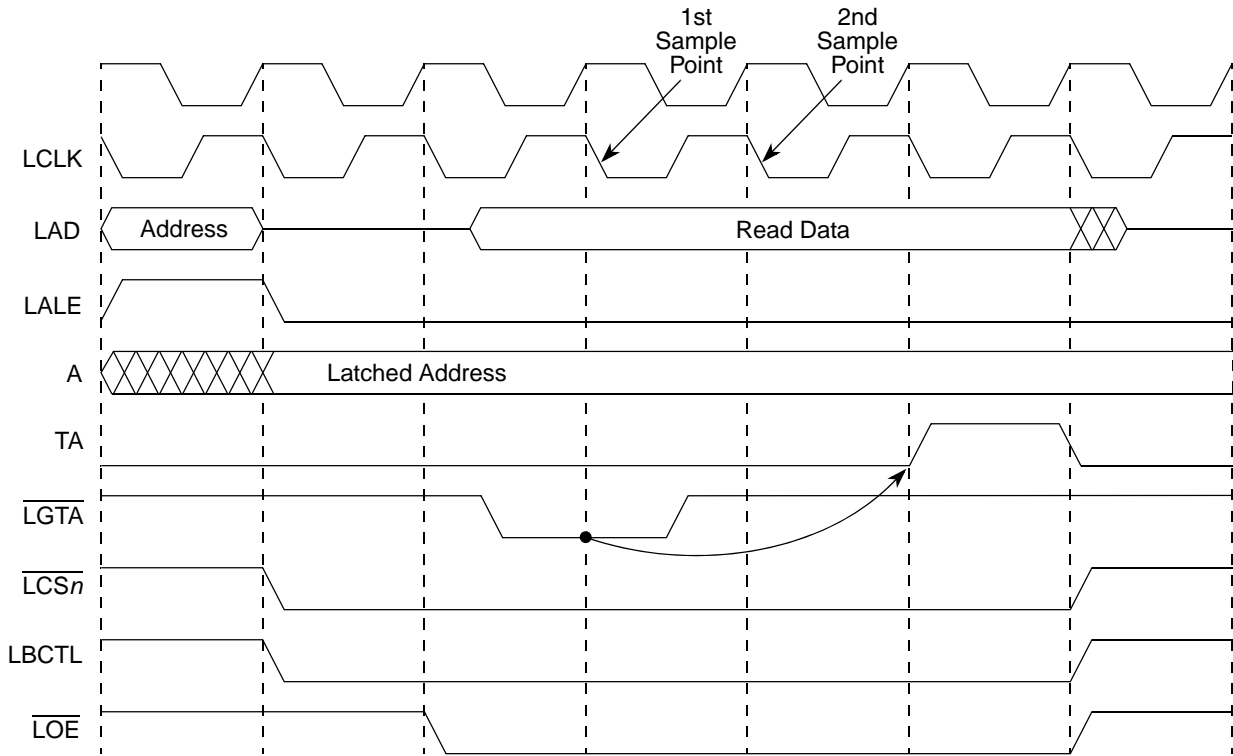


Figure 11-41. External Termination of GPCM Access(PLL Bypass Mode)

11.4.2.5 GPCM Boot Chip-Select Operation

Boot chip-select operation allows address decoding for a boot ROM before system initialization. $\overline{LCS0}$ is the boot chip-select output; its operation differs from other external chip-select outputs after a system reset. When the core begins accessing memory after system reset, $\overline{LCS0}$ 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. $\overline{LCS0}$ 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 11-32 describes the initial values of the boot bank in the memory controller.

Table 11-32. Boot Bank Field Values after Reset for GPCM as Boot Controller

Register	Field	Setting
BR0	BA	0000_0000_0000_0000_0
	PS	From RCWH[ROMLOC]
	DECC	00
	WP	0
	MSEL	000
	—	—
	V	1
OR0	AM	0000_0000_0000_0000_0
	BCTLD	0
	CSNT	1
	ACS	11
	XACS	1
	SCY	1111
	SETA	0
	TRLX	1
	EHTR	1
	EAD	1

11.4.3 Flash Control Machine (FCM)

The FCM provides a glueless interface to parallel-bus NAND Flash EEPROM devices. The FCM contains three basic configuration register groups—BR_n, OR_n, and FMR.

Figure 11-42 shows a simple connection between an 8-bit port size NAND Flash EEPROM and the eLBC in FCM mode. Commands, address bytes, and data are all transferred on LAD[0:7]¹, with $\overline{\text{LFW}}\overline{\text{E}}$ asserted for transfers written to the device, or $\overline{\text{LFR}}\overline{\text{E}}$ asserted for transfers read from the device. eLBC signals LFCLE and LFALE determine whether writes are of type command (only LFCLE asserted), address (only LFALE asserted), or write data (neither LFCLE nor LFALE asserted). The NAND Flash RDY/ $\overline{\text{BSY}}$ pin is normally open-drain, and should be pulled high by a 4.7-K Ω resistor. On system reset, a global (boot)

1. Note bit numbering reversal: LAD[0] (msb) connects to Flash IO[7], while LAD[7] (lsb) connects to IO[0].

chip-select is available that provides a boot ROM chip-select ($\overline{\text{LCS0}}$) prior to the system being fully configured.

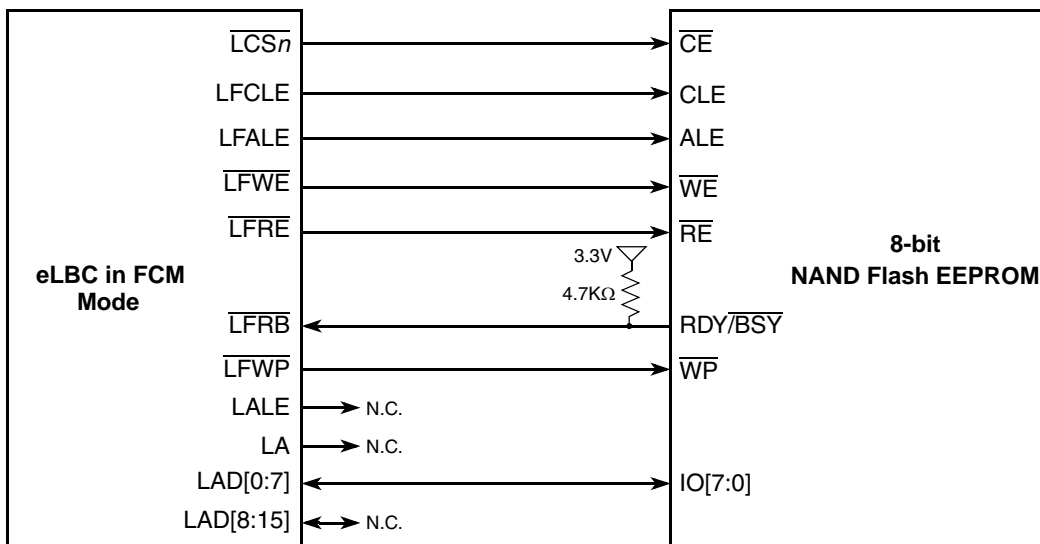


Figure 11-42. Local Bus to 8-Bit FCM Device Interface

Basic read access timing for FCM is shown in Figure 11-43. Although LCLK is shown for reference, NAND Flash EEPROMs do not make use of the clock.

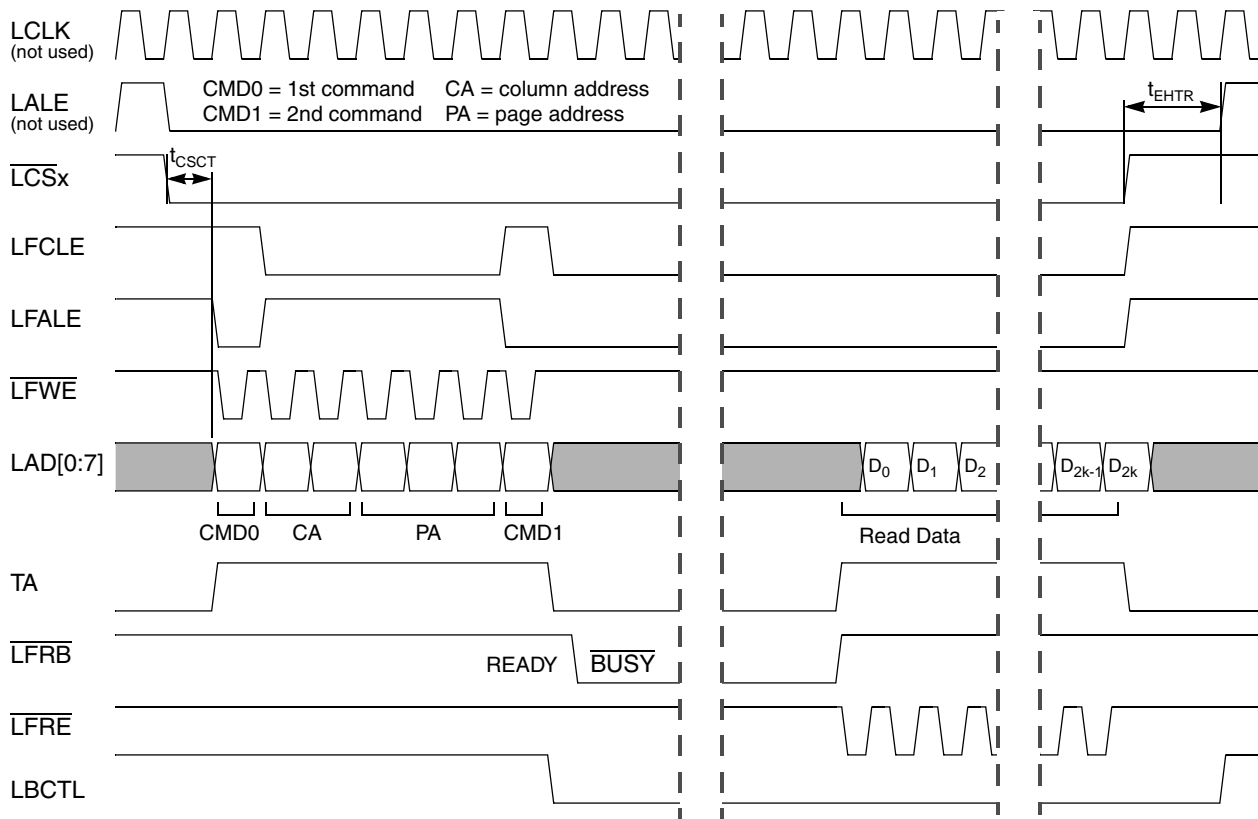


Figure 11-43. FCM Basic Page Read Timing
(PGS = 1, CSCT = 0, CST = 0, CHT = 1, RST = 1, SCY = 0, TRLX = 0, EHTR = 1)

Following the assertion of LALE, FCM asserts \overline{LCSn} to commence a command sequence to the Flash device. After a delay of t_{CSCT} , the first command can be written to the device on assertion of \overline{LFWE} , followed by any parameters (typically address bytes and data), and concluded with a secondary command. In many cases, the second command initiates a long-running operation inside the Flash device, which pulls the wired-OR pin \overline{LFRB} low to indicate that the device is busy. Since in Figure 11-43 FCM is now expecting a read response, it takes LBCTL low to turnaround any bus transceivers that are present. Upon \overline{LFRB} indicating ready status, FCM asserts \overline{LFRE} repeatedly to recover bytes of read data, and the bytes are stored in eLBC's FCM buffer RAM while an ECC is optionally computed on the bytes transferred. Finally, FCM negates \overline{LCSn} and delays eLBC by t_{EHTR} before any subsequent memory access occurs.

11.4.3.1 FCM Buffer RAM

Read and write accesses to eLBC banks controlled by FCM do not access attached NAND Flash EEPROMs directly. Rather, these accesses read and write the FCM buffer RAM—a single, shared 8-Kbyte space internal to eLBC and mapped by the base address of every FCM bank. Even though each FCM-controlled bank will have a different base address to differentiate it, all accesses to such banks will access the same buffer space. External eLBC signals, such as LALE and \overline{LCSn} , will not assert upon accesses to the buffer RAM. The FCM buffer RAM is logically divided into two or more buffers,

depending on the setting of $ORn[PGS]$, with different buffers being accessible concurrently by software and FCM.

To perform a page read operation from a NAND Flash device, software initializes the FCM command, mode, and address registers, before issuing a special operation (FMR[OP] set non-zero) to a particular FCM-controlled bank. FCM will execute the sequence of op-codes held in FIR, reading data from the Flash device into the shared buffer RAM. While this read is taking place, software is free to access any data stored in other, currently inactive buffers of the FCM buffer RAM through reads or writes to any bank controlled by FCM. If command completion interrupts are enabled, an interrupt will be generated once FCM has completed the read. When FCM has completed its last command, software can switch to the newly read buffer and issue further commands.

To perform a page write operation, software first prepares data to be written in a fresh buffer. Then, the FCM command, mode, and address registers are initialized, and a special operation (FMR[OP] set non-zero) is issued to a particular FCM-controlled bank. FCM will execute the sequence of op-codes held in FIR, writing data from shared buffer RAM to the Flash device. To ensure that the device is enabled for programming, software must initialize $FMR[OP] = 11$, which prevents assertion of \overline{LFWP} during the write. While this write is taking place, software is free to access any data stored in other, currently inactive buffers of the FCM buffer RAM through reads or writes to any bank controlled by FCM. When FCM has completed its last command, software can re-use the previously written buffer and issue further commands.

See [Section 11.4.3.4.2, “Boot Block Loading into the FCM Buffer RAM,”](#) for a description of the shared buffer RAM layout during boot.

11.4.3.1.1 Buffer Layout and Page Mapping for Small-Page NAND Flash Devices

The FCM buffer space is divided into eight 1-Kbyte buffers for small-page devices ($ORn[PGS] = 0$), mapped as shown in [Figure 11-44](#). Each page in a small-page NAND Flash comprises 528 bytes, where 512 bytes appear as main region data, and 16 bytes appear as spare region data. The EEPROM's page numbered P is associated with buffer number $(P \bmod 8)$, where $P = FPAR[PI]$. Since the bank size set by $ORn[AM]$ will be greater than 8 Kbytes, an identical image of the FCM buffer RAM appears replicated every 8 Kbytes throughout the bank address space. It is recommended that the bank size be set to 32 Kbytes, which covers a single NAND Flash block for small-page devices.

For FCM commands, register FPAR sets the page address and, therefore, also the buffer number. In the case that $FBCR[BC] = 0$, FCM transfers an entire page, comprising the 512-byte main region followed by the 16-byte spare region; the 496-byte reserved region is not accessed, and remains undefined for software. However, for commands given a specific byte count in $FBCR[BC]$, $FPAR[MS]$ locates the starting address in either the main region ($MS = 0$) or the spare region ($MS = 1$). Where different eLBC banks control both

small and large-page devices, a large-page 4-Kbyte buffer must be assigned to either the first 4 or last 4 small-page buffers.

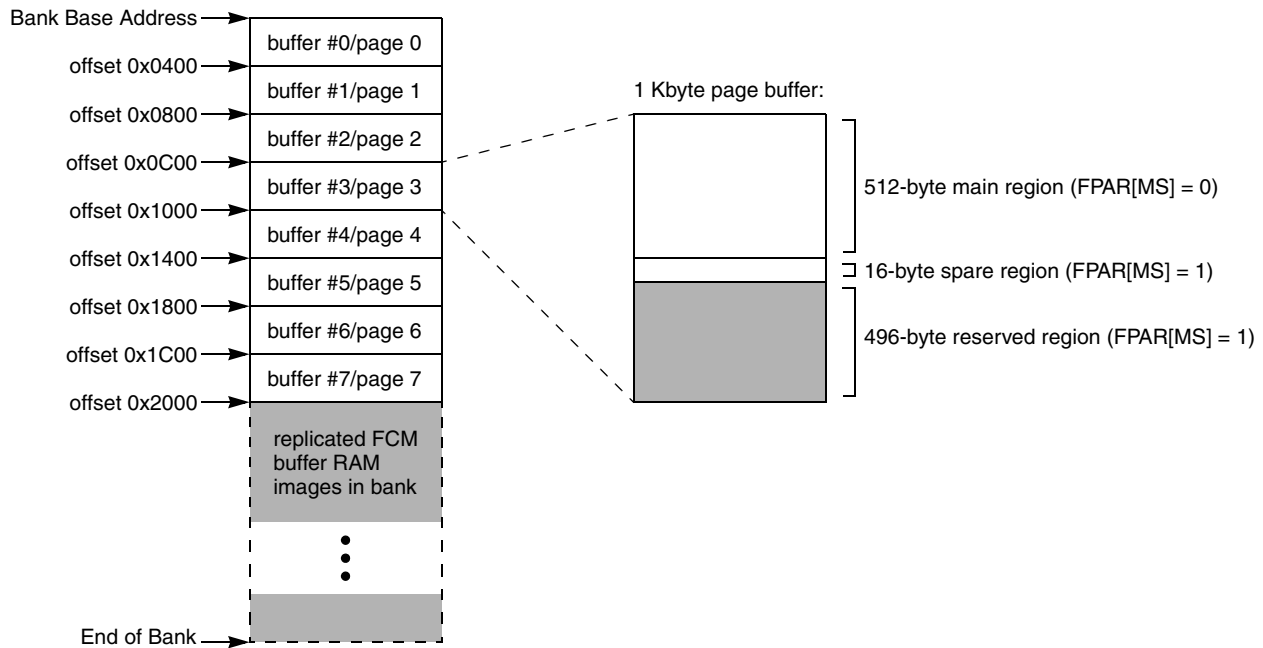


Figure 11-44. FCM Buffer RAM Memory Map for Small-Page (512-byte page) NAND Flash Devices

11.4.3.1.2 Buffer Layout and Page Mapping for Large-Page NAND Flash Devices

The FCM buffer space is divided into two 4 Kbyte buffers for large-page devices ($ORn[PGS] = 1$), mapped as shown in Figure 11-45. Each page in a large-page NAND Flash comprises 2112 bytes, where 2048 bytes appear as main region data, and 64 bytes appear as spare region data. The EEPROM's page numbered P is associated with buffer number $(P \bmod 2)$, where $P = FPAR[PI]$. Since the bank size set by $ORn[AM]$ will be greater than 8 Kbytes, an identical image of the FCM buffer RAM appears replicated every 8 Kbytes throughout the bank address space. It is recommended that the bank size be set to 256 Kbytes, which covers a single NAND Flash block for large-page devices.

For FCM commands, register FPAR sets the page address and, therefore, also the buffer number. In the case that $FBCR[BC] = 0$, FCM transfers an entire page, comprising the 2048-byte main region followed by the 64-byte spare region; the 1984-byte reserved region is not accessed, and remains undefined for software. However, for commands given a specific byte count in $FBCR[BC]$, $FPAR[MS]$ locates the starting address in either the main region ($MS = 0$) or the spare region ($MS = 1$). Where different eLBC

banks control both small and large-page devices, a large-page 4 Kbyte buffer must be assigned to either the first 4 or last 4 small-page buffers.

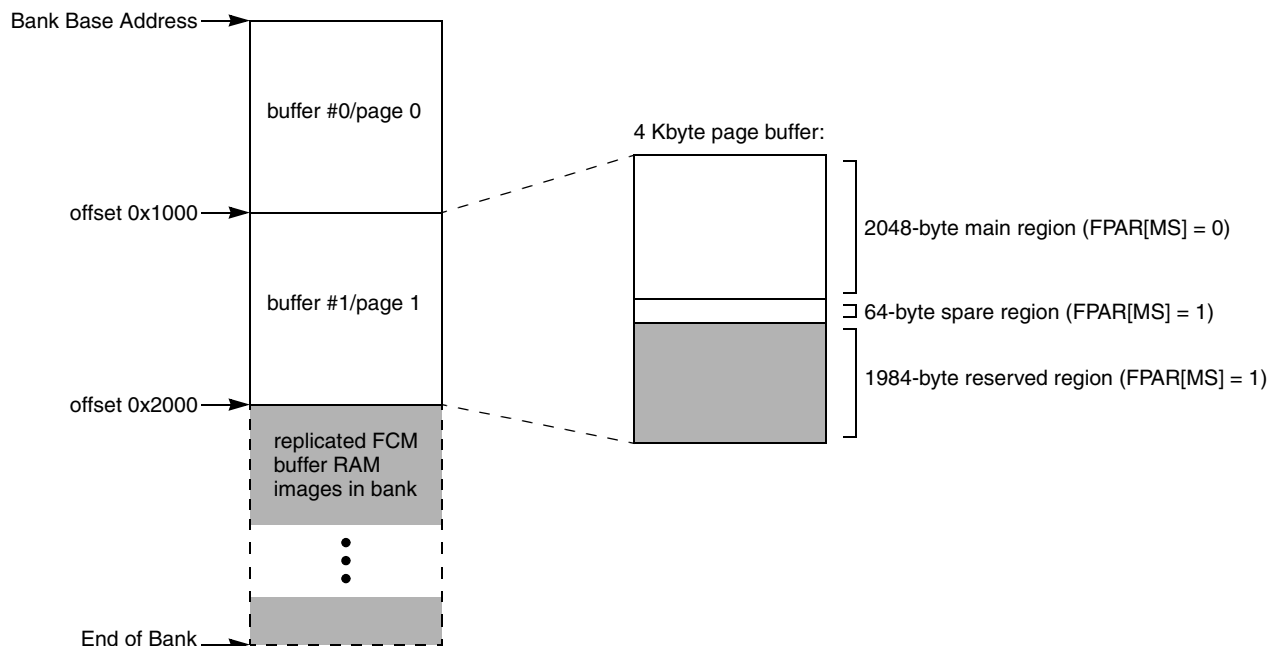


Figure 11-45. FCM Buffer RAM Memory Map for Large-Page (2-Kbyte page) NAND Flash Devices

11.4.3.1.3 Error Correcting Codes and the Spare Region

The FCM’s ECC engine makes use of data in the NAND Flash spare region to store pre-computed ECC code words. ECC is calculated in a single pass over blocks of 512 bytes of data in the main region. The setting of FMR[ECCM] determines the location of the 24-bit ECC in the spare region.

The basic ECC algorithm is depicted in Figure 11-46. The stream of data bytes is considered to form a matrix having 8 columns (corresponding with the device bus IO[7:0] or IO[15:8]) and 512 rows (corresponding with each byte in the ECC block).

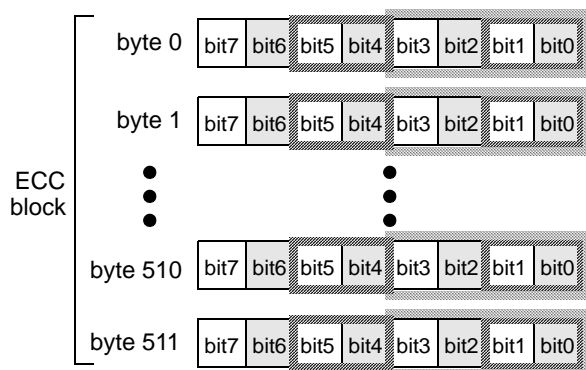


Figure 11-46. FCM ECC Calculation

The placement of ECC code words in relation to FMR[ECCM] is shown in Figure 11-47. For small-page devices, only a single 512-byte main region is ECC-protected. For large-page devices, there are four

adjacent main regions, and each has a 16-byte spare region—of which only one is shown in the figure. If eLBC is configured to generate ECC ($BR_n[DECC] = 10$), FCM will substitute on full-page write transfers the three code word bytes in place of the spare region data originally provided at the locations shown in Figure 11-47. Transfers shorter than a full page, however, require software to prepare the appropriate ECC in the spare region. Similarly, FCM can check and correct bit errors on full-page reads if $BR_n[DECC] = 01$ or 10 . A correctable error is a single bit error in any 512-byte block of main region data, as judged by comparison of a regenerated ECC with the ECC retrieved from the spare region, or a single bit error in the retrieved ECC only. Bit errors in the main region are corrected before FCM completes its final read transfer and signals an event in $LTESR[CC]$. Errors that appear more complex (two or more bits in error per 512-byte block) are not corrected, but are flagged as parity errors by FCM. The bit vector in $LTEATR[PB]$ can be checked to determine which 512-byte blocks in a large-page NAND Flash main region were found to be non-correctable.

ECCM	Byte 0	Byte 511	Other Mains	Spare 0	5	6	7	8	9	10	11	12	13	14	15
0	Main Region			—	EC0	EC1	EC2	—							
1	Main Region			—				EC0	EC1	EC2	—				

Figure 11-47. ECC Placement in NAND Flash Spare Regions in Relation to $FMR[ECCM]$

11.4.3.2 Programming FCM

FCM has a fully general command and data transfer sequencer that caters for both common and specific/proprietary NAND Flash command sequences. The command sequencer reads a program out of the FIR register, which can hold up to 8 instructions, each represented by a 4-bit op-code, as illustrated in Figure 11-48. The first instruction executed is read from $FIR[OP0]$, the next is read from $FIR[OP1]$, and likewise to subsequent instructions, ending at $FIR[OP7]$ or until the only instructions remaining are NOPs. If FIR contains nothing but NOP instructions, FCM will not assert \overline{LCSn} , otherwise, \overline{LCSn} is asserted prior to the first instruction and remains asserted until the last instruction has completed. If $LTESR[CC]$ is enabled, completion of the last instruction will trigger a command completion event interrupt from eLBC.

Prior to executing a sequence, necessary operands for the instructions will need to be set in the FMR, FCR, MDR, FBCR, FBAR, and FPAR registers. The AS0–AS3 address and data pointers associated with FCM’s use of MDR all reset to select AS0 at the start of the instruction sequence. A complete list of op-codes can be found in Section 11.3.1.17, “Flash Instruction Register (FIR).”

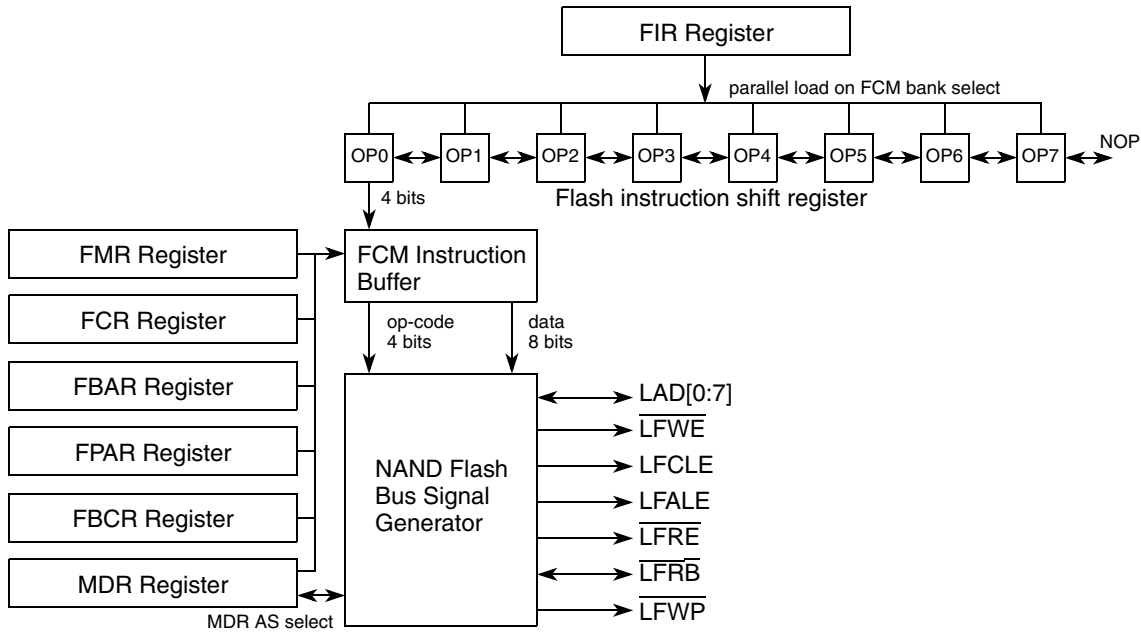


Figure 11-48. FCM Instruction Sequencer Mechanism

11.4.3.2.1 FCM Command Instructions

There are two kinds of command instruction:

- Commands that issue immediately—CM0, CM1, CM2, and CM3. These commands write a single command byte by asserting LFCLE and $\overline{\text{LFWE}}$ while driving an 8-bit command onto LAD[0:7]. Op-code CM n sources its command byte from field FCR[CMD n], therefore up to four different commands can be issued in any FCM instruction sequence.
- Commands that wait for $\overline{\text{LFRB}}$ to be sampled high (EEPROM in ready state) before issuing—CW0, and CW1. These commands first poll the $\overline{\text{LFRB}}$ pin, waiting for it to go high, before writing a single command byte onto LAD[0:7], sourced from FCR[CMD n] for op-code CW n . It is necessary to use CW n op-codes whenever the EEPROM is expected to be in a busy state (such as following a page read, block erase, or program operation) and therefore initially unresponsive to commands. To avoid deadlock in cases where the device is already available, FCM does not expect a transition on $\overline{\text{LFRB}}$. Rather, FCM waits for $8 \times (2 + \text{OR}_n[\text{SCY}])$ clock cycles (when $\text{OR}_n[\text{TRLX}] = 0$) or $16 \times (2 + \text{OR}_n[\text{SCY}])$ clock cycles (when $\text{OR}_n[\text{TRLX}] = 1$) before sampling the level of $\overline{\text{LFRB}}$. If the level of $\overline{\text{LFRB}}$ does not return high before a time-out set by FMR[CWTO] occurs, FCM proceeds to issue the command normally, and a FCT event is issued to LTESR.

The manufacturer’s datasheet should be consulted to determine values for programming into the FCR register, and whether a given command in the sequence is expected to initiate busy device behavior.

11.4.3.2.2 FCM No-Operation Instruction

A NOP instruction that appears in FIR ahead of the last instruction is executed with the timing of a regular command instruction, but neither LFCLE nor $\overline{\text{LFW}}\overline{\text{E}}$ are asserted. Thus a NOP instruction may be used to insert a pause matching the time taken for a regular command write.

11.4.3.2.3 FCM Address Instructions

Address instructions are used to issue addresses to the NAND Flash EEPROM. A complete device address is formed from a sequence of one or more bytes, each written onto LAD[0:7] with LFALE and $\overline{\text{LFW}}\overline{\text{E}}$ asserted together. There are three kinds of address generation provided:

- Column address—CA. A column address comprises one byte ($\text{OR}_n[\text{PGS}] = 0$) or two bytes ($\text{OR}_n[\text{PGS}] = 1$) locating the starting byte or word to be transferred in the next page read or write sequence. FPAR[CI] sets the value of the column index provided that FBCR[BC] is non-zero. In the case that FBCR[BC] = 0, a column index of zero is issued to the device, regardless of the value in FPAR[CI].
- Page address—PA. A page address comprises 2, 3, or 4 bytes, depending on the setting of FMR[AL], and locates the data page in the NAND Flash address space. The complete page address is the concatenation of the block index, read from FBAR[BLK], with the page-in-block index, read from FPAR[PI]. The page address length set in FMR[AL] should correspond with the size of EEPROM being accessed. Similarly, the block index in FBAR[BLK] must not exceed the maximum block index for the device, as most devices require reserved address bits to be written as zero.
- User-defined address—UA. This instruction allows the FCM to write a user-defined address byte, which is read from the next AS field in MDR, starting at MDR[AS0]. Each subsequent UA instruction reads an adjacent AS field in MDR, until all four AS bytes (MDR[AS0], MDR[AS1], MDR[AS2], MDR[AS3]) have been sent; a fifth and any following UA instructions send zero as the address byte. Note that each UA instruction advances the MDR pointer for writes by one byte, and therefore a mix of UA and WS instructions can consume adjacent bytes from MDR.

11.4.3.2.4 FCM Data Read Instructions

Data read instructions assert $\overline{\text{LFR}}\overline{\text{E}}$ repeatedly to transfer one or more bytes of read data from the NAND Flash EEPROM. Data read instructions are distinguished by their data destination:

- Read data to buffer RAM immediately—RB. This instruction reads FBCR[BC] bytes of data into the current FCM RAM buffer addressed by FPAR. If FBCR[BC] = 0, an entire page (including spare region) is transferred in a burst, starting at the page boundary, and the ECC calculation is checked against the ECC stored in the spare region. Correctable ECC errors are corrected; other errors may cause an interrupt if enabled. If the value of FBCR[BC] takes the read pointer beyond the end of the spare region in the buffer, FCM discards any excess bytes read.
- Read data/status to MDR immediately—RS. This instruction asserts $\overline{\text{LFR}}\overline{\text{E}}$ exactly once to read one byte (8-bit port size) of data into the next AS field of MDR. Reads beyond the fourth byte of MDR are discarded. The MDR read pointer is independent of the MDR write pointer used by UA and WS instructions.

- Read data to buffer RAM once waited on ready—RBW. This instruction first polls the $\overline{\text{LFRB}}$ pin, waiting for it to go high, before proceeding with a read to buffer as described for the RB instruction. Sampling and time-outs for polling the $\overline{\text{LFRB}}$ pin follow the behavior of CW_n instructions.
- Read data/status to MDR once waited on ready—RSW. This instruction first polls the $\overline{\text{LFRB}}$ pin, waiting for it to go high, before proceeding with a status read to MDR as described for the RS instruction. Sampling and time-outs for polling the $\overline{\text{LFRB}}$ pin follow the behavior of CW_n instructions.

11.4.3.2.5 FCM Data Write Instructions

Data write instructions assert $\overline{\text{LFWE}}$ repeatedly (with LFCLE and LFALE both negated) to transfer one or more bytes of write data to the NAND Flash EEPROM. Data write instructions are distinguished by their data source:

- Write data from FCM buffer RAM—WB. This instruction writes $\text{FBCR}[\text{BC}]$ bytes of data from the current FCM RAM buffer addressed by FPAR . If $\text{FBCR}[\text{BC}] = 0$, an entire page (including spare region) is transferred in a burst, starting at the page boundary, and the ECC calculation is stored in the and spare region in accordance with the setting of $\text{FMR}[\text{ECCM}]$. If the value of $\text{FBCR}[\text{BC}]$ takes the write pointer beyond the end of the spare region in the buffer, the value of data written by FCM is undefined.
- Write data/status from MDR—WS. This instruction asserts $\overline{\text{LFWE}}$ exactly once to write one byte (8-bit port size) of data taken from the next AS field of MDR. Attempts to write beyond four bytes of MDR has the effect of writing zeros. The MDR write pointer is independent of the MDR read pointer used by RS and RSW instructions.

11.4.3.3 FCM Signal Timing

If $\text{BR}_n[\text{MSEL}]$ selects the FCM, the attributes for the memory cycle are taken from OR_n . These attributes include the CSCT , CST , CHT , RST , SCY , TRLX , and EHTR fields.

11.4.3.3.1 FCM Chip-Select Timing

The timing of $\overline{\text{LCS}}_n$ assertion in FCM mode is illustrated by the timing diagram in [Figure 11-43](#). $\overline{\text{LCS}}_n$ is asserted immediately following LALE negation, and remains asserted until the last instruction in FIR has completed. The delay, t_{CSCT} , between $\overline{\text{LCS}}_n$ assertion and commencement of the first NAND Flash instruction is controlled by $\text{OR}_n[\text{CSCT}]$ and $\text{OR}_n[\text{TRLX}]$, as shown in [Table 11-33](#). $\text{OR}_n[\text{CSCT}]$ should be set in accordance with the NAND Flash EEPROM chip-select to $\overline{\text{WE}}$ set-up time specification.

Table 11-33. FCM Chip-Select to First Command Timing

$\text{OR}_n[\text{TRLX}]$	$\text{OR}_n[\text{CSCT}]$	$\overline{\text{LCS}}_n$ to First Command Delay
0	0	1 LCLK clock cycle
0	1	4 LCLK clock cycles
1	0	2 LCLK clock cycles
1	1	8 LCLK clock cycles

11.4.3.3.2 FCM Command, Address, and Write Data Timing

The FCM command (CM0–CM3, CW0, CW1), address (CA, PA, UA), and data write (WB, WS) instructions all share the same basic timing attributes. Assertion of $\overline{\text{LFW\!E}}$ initiates transfer via LAD[0:7], and the options in OR_n for FCM mode establish the set-up, hold, and wait state timings with respect to $\overline{\text{LFW\!E}}$, as shown in Figure 11-49.

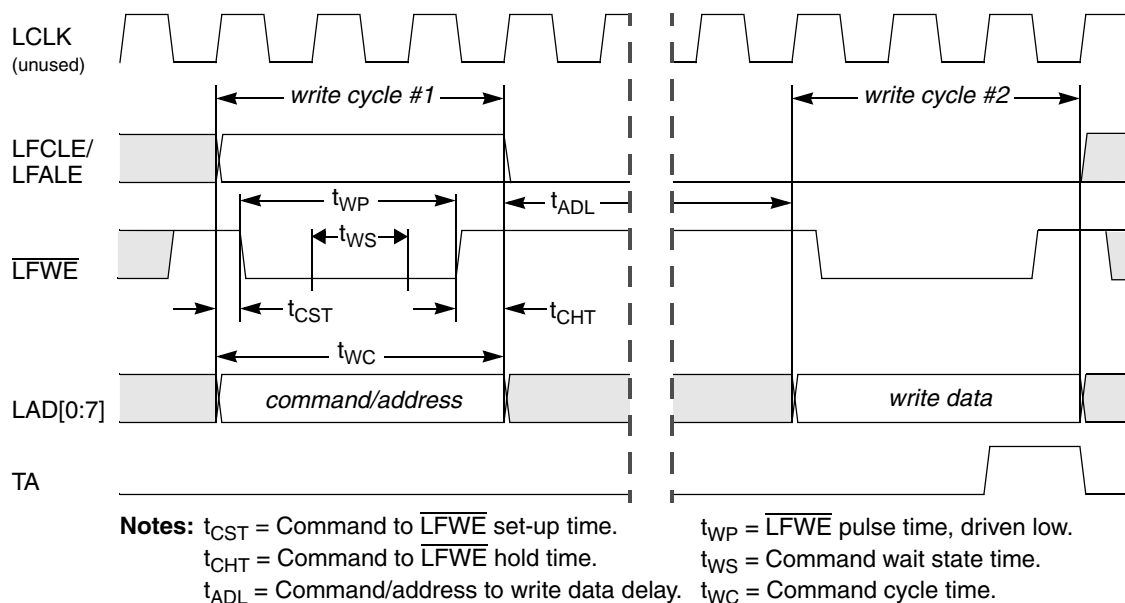


Figure 11-49. Timing of FCM Command/Address and Write Data Cycles
 (for $\text{TRLX} = 0$, $\text{CHT} = 0$, $\text{CST} = 1$, $\text{SCY} = 1$, $\text{CLKDIV} = 4 \times \text{N}$)

The timing parameters are summarized in Table 11-34.

Table 11-34. FCM Command, Address, and Write Data Timing Parameters

Option Register Attributes			Timing Parameter (LCLK Clock Cycles) ¹					
TRLX	CHT	CST	t_{CST}	t_{CHT}	t_{WS}	t_{WP}	t_{WC}	t_{ADL}
0	0	0	0	$\frac{1}{2}$	SCY	$1\frac{1}{2} + \text{SCY}$	$2 + \text{SCY}$	$4 \times (2 + \text{SCY})$
0	0	1	$\frac{1}{4}$	$\frac{1}{2}$	SCY	$1\frac{1}{4} + \text{SCY}$	$2 + \text{SCY}$	$4 \times (2 + \text{SCY})$
0	1	0	0	1	SCY	$1 + \text{SCY}$	$2 + \text{SCY}$	$4 \times (2 + \text{SCY})$
0	1	1	$\frac{1}{4}$	1	SCY	$\frac{3}{4} + \text{SCY}$	$2 + \text{SCY}$	$4 \times (2 + \text{SCY})$
1	0	0	$\frac{1}{2}$	$1\frac{1}{2}$	$2 \times \text{SCY}$	$1 + 2 \times \text{SCY}$	$3 + 2 \times \text{SCY}$	$8 \times (2 + \text{SCY})$
1	0	1	1	$1\frac{1}{2}$	$2 \times \text{SCY}$	$\frac{1}{2} + 2 \times \text{SCY}$	$3 + 2 \times \text{SCY}$	$8 \times (2 + \text{SCY})$
1	1	0	$\frac{1}{2}$	2	$2 \times \text{SCY}$	$\frac{1}{2} + 2 \times \text{SCY}$	$3 + 2 \times \text{SCY}$	$8 \times (2 + \text{SCY})$
1	1	1	1	2	$2 \times \text{SCY}$	$2 \times \text{SCY}$	$3 + 2 \times \text{SCY}$	$8 \times (2 + \text{SCY})$

¹ In the parameters, SCY refers to a delay of $\text{OR}_n[\text{SCY}]$ clock cycles.

An example of minimum delay command timing appears in [Figure 11-50](#). Note that the set-up, wait-state, and hold timing of command, address, and write data cycles with respect to $\overline{\text{LFW\!E}}$ assertion are all identical, and that the minimum cycle extends for two LCLK clock cycles.

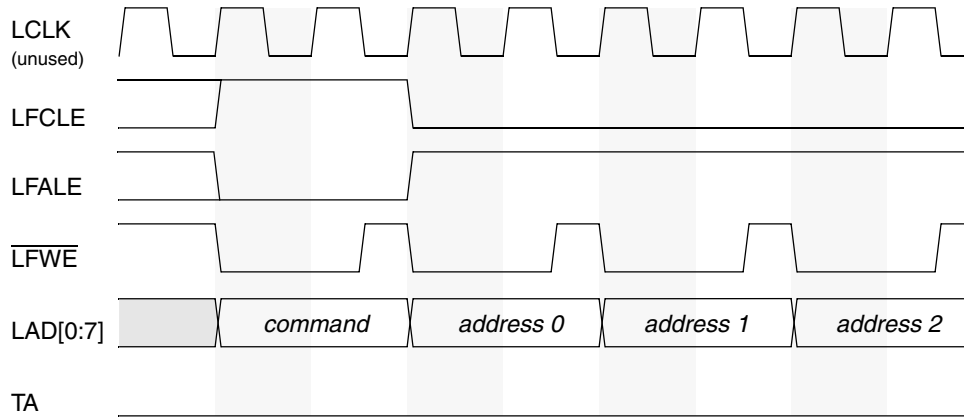


Figure 11-50. Example of FCM Command and Address Timing with Minimum Delay Parameters (for $\text{TRLX} = 0$, $\text{CHT} = 0$, $\text{CST} = 0$, $\text{SCY} = 0$, $\text{CLKDIV} = 4 \cdot \text{N}$)

An example of relaxed command timing is shown in [Figure 11-51](#).

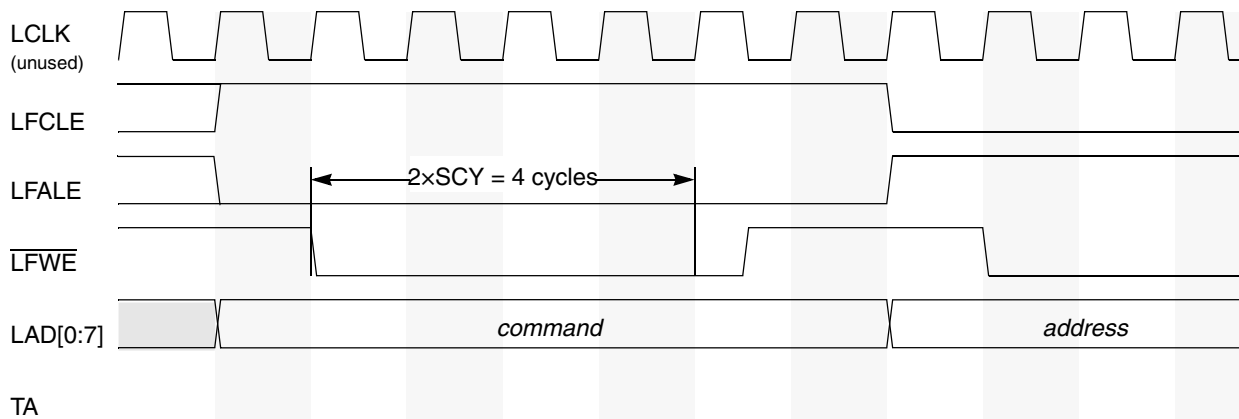


Figure 11-51. Example of FCM Command and Address Timing with Relaxed Parameters (for $\text{TRLX} = 1$, $\text{CHT} = 0$, $\text{CST} = 1$, $\text{SCY} = 2$, $\text{CLKDIV} = 4 \cdot \text{N}$)

11.4.3.3.3 FCM Ready/Busy Timing

Instructions CW0, CW1, RBW, and RSW force FCM to observe the state of the $\overline{\text{LFRB}}$ pin, which may be driven low by a long-latency NAND Flash operation, such as a page read. Following the issue of such commands, FCM waits as shown in [Figure 11-52](#) before sampling the state of $\overline{\text{LFRB}}$. This guards against observing $\overline{\text{LFRB}}$ before it has been properly driven low by the device, but does not preclude $\overline{\text{LFRB}}$ from

remaining high after a command. In addition, FCM samples and compares the state of $\overline{\text{LFRB}}$ on two consecutive cycles of LCLK to filter out noise on this signal as it rises to the ready state ($\overline{\text{LFRB}} = 1$).

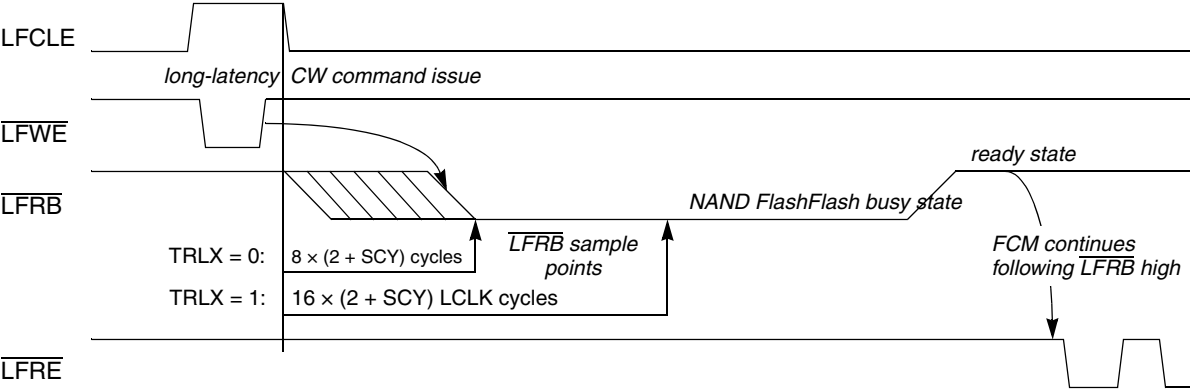
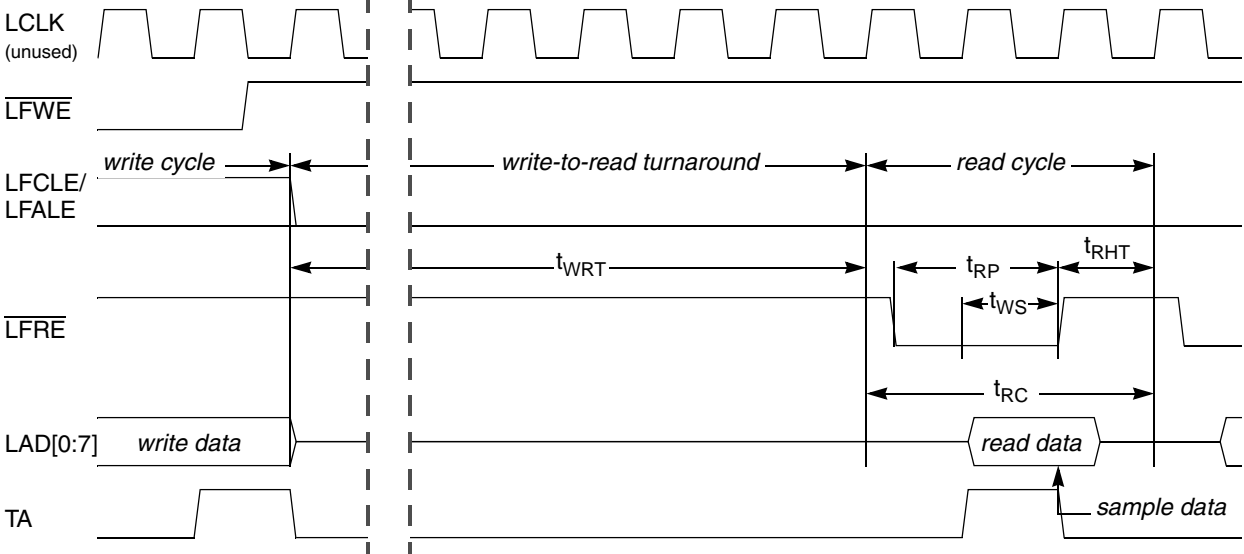


Figure 11-52. FCM Delay Prior to Sampling $\overline{\text{LFRB}}$ State

11.4.3.3.4 FCM Read Data Timing

The timing for read data transfers is shown in Figure 11-53. Upon assertion of $\overline{\text{LFRE}}$, the Flash device will enable its output drivers and drive valid read data while $\overline{\text{LFRE}}$ is held low. FCM samples read data on the rising edge of $\overline{\text{LFRE}}$, which follows an optional number of wait states. Note that FCM will delay the first read if a RBW or RSW instruction is issued, in which case $\overline{\text{LFRB}}$ sample timing takes effect (see Section 11.4.3.3.3, “FCM Ready/Busy Timing”).



- Notes: t_{RP} = $\overline{\text{LFRE}}$ pulse time, read period. t_{WS} = Read wait state time.
- t_{RHT} = $\overline{\text{LFRE}}$ hold time. t_{RC} = Read data cycle time.
- t_{WRT} = Write to read turnaround time.

Figure 11-53. FCM Read Data Timing (for $\text{TRLX} = 0, \text{RST} = 0, \text{SCY} = 1, \text{CLKDIV} = 4 \cdot N$)

The timing parameters are summarized in [Table 11-35](#).

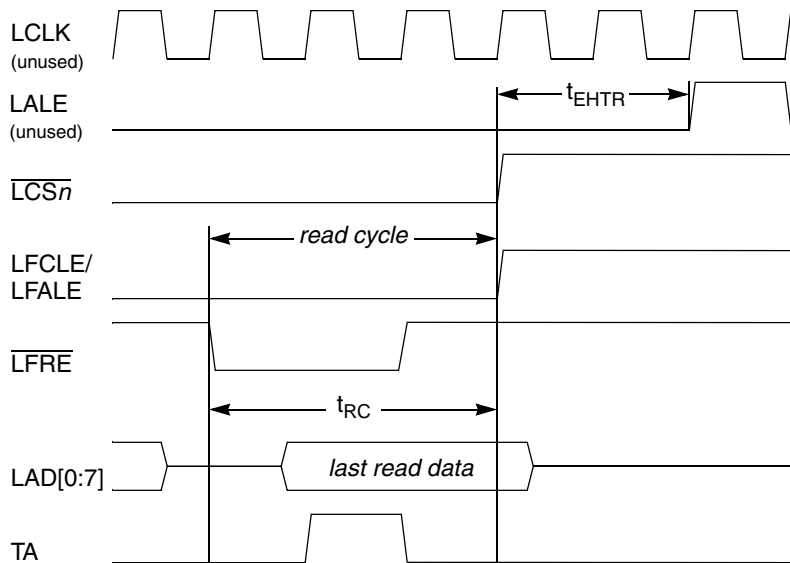
Table 11-35. FCM Read Data Timing Parameters

Option Register Attributes		Timing Parameter (LCLK Clock Cycles) ¹				
TRLX	RST	t_{RP}	t_{RHT}	t_{WS}	t_{RC}	t_{WRT}
0	0	$\frac{3}{4} + SCY$	1	SCY	$2 + SCY$	$4 \times (2 + SCY)$
0	1	$1 + SCY$	1	SCY	$2 + SCY$	$4 \times (2 + SCY)$
1	0	$\frac{1}{2} + 2 \times SCY$	2	$2 \times SCY$	$3 + 2 \times SCY$	$8 \times (2 + SCY)$
1	1	$1 + 2 \times SCY$	2	$2 \times SCY$	$3 + 2 \times SCY$	$8 \times (2 + SCY)$

¹ In the parameters, SCY refers to a delay of $ORn[SCY]$ clock cycles.

11.4.3.3.5 FCM Extended Read Hold Timing

Allowance for slow output driver turn-off when reading NAND Flash EEPROMs is made via setting of $ORn[EHTR]$ and $ORn[TRLX]$. The extended read data hold time, shown at t_{EHTR} in [Figure 11-43](#) and [Figure 11-54](#), is a delay inserted by FCM between the last data read and another eLBC memory access (requiring LALE assertion). \overline{LCSn} is negated during t_{EHTR} to allow external devices and bus transceivers time to disable their drivers.



Notes: t_{RC} = Read data cycle time.
 t_{EHTR} = Extended read data hold time.

Figure 11-54. FCM Read Data Timing with Extended Hold Time
 (for $TRLX = 0$, $EHTR = 1$, $RST = 1$, $SCY = 1$, $CLKDIV = 4 \times N$)

11.4.3.4 FCM Boot Chip-Select Operation

Boot chip-select operation allows address decoding for a boot ROM before system initialization. $\overline{LCS0}$ is the boot chip-select output; its operation differs from other external chip-select outputs after a system reset.

When the core begins accessing memory after system reset, $\overline{LCS0}$ is asserted initially to load a 4-Kbyte boot block into the FCM buffer RAM, but core instruction fetches occur from the buffer RAM.

11.4.3.4.1 FCM Bank 0 Reset Initialization

The boot chip-select also provides a programmable port size, which is configured during reset. The boot chip-select does not provide write protection. $\overline{LCS0}$ 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 11-36 describes the initial values of the boot bank in the memory controller.

Table 11-36. Boot Bank Field Values after Reset for FCM as Boot Controller

Register	Field	Setting
BR0	BA	0000_0000_0000_0000_0
	PS	From 01
	DECC	00
	WP	0
	MSEL	001
	V	0
OR0	AM	0000_0000_0000_0000_0
	BCTLD	0
	PGS	From RCWH[ROMLOC]
	CSCT	1
	CST	1
	CHT	1
	RST	1
	TRLX	1
	EHTR	1

11.4.3.4.2 Boot Block Loading into the FCM Buffer RAM

If FCM is selected as the boot ROM controller from power-on-reset configuration, eLBC will automatically load from bank 0 a single 4 Kbyte page of boot code into the FCM buffer RAM during \overline{HRESET} . The CPU can execute boot code directly from the FCM buffer RAM, but must ensure that any further data read from the NAND Flash EEPROM is transferred under software control in order to continue the bootstrap process.

Since OR0[AM] is initially cleared during reset, all CPU fetches to eLBC will access the FCM buffer RAM, which appears in the memory map as a 4-Kbyte RAM. No NAND Flash spare regions are mapped during boot, therefore only 4 Kbytes of contiguous, main region data, loaded from the first pages of the

boot block, are accessible in eLBC bank 0, as indicated in [Figure 11-55](#).

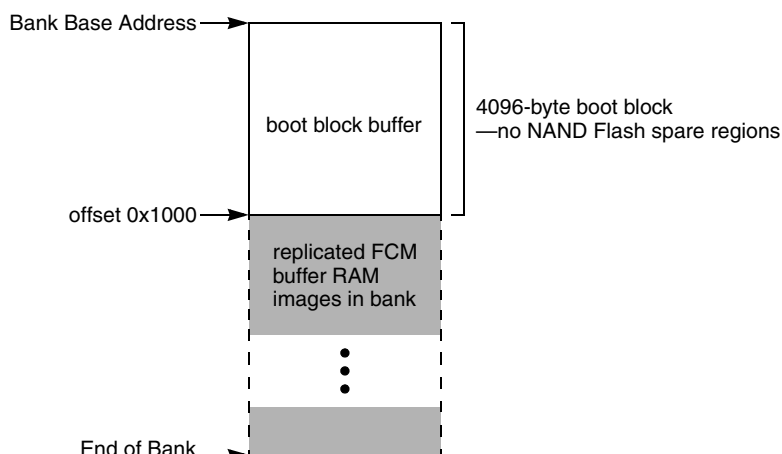


Figure 11-55. FCM Buffer RAM Memory Map During Boot Loading

The process for booting is as follows:

1. Following negation of $\overline{\text{PORESET}}$, eLBC is released from reset and commences automatic boot block loading if FCM is selected as the boot ROM location. Small-page or large-page, 8-bit NAND Flash devices can be used for boot loading when enabled with $\overline{\text{LCS0}}$. eLBC drives $\overline{\text{LFWP}}$ low during boot accesses to prevent accidental erasure of the NAND Flash boot ROM.
2. FCM starts searching for a valid boot block at block index 0.
3. FCM reads the spare regions of the first two pages of the current block, checking the bad block indication (BI) bytes to validate the block for reading. BI bytes must all hold the value 0xFF for the page to be considered readable.
 - For small-page devices, BI is a single byte read from spare region byte offset 5.
 - For large-page devices, BI is a single byte read from spare region byte offset 0.

If either of the first two pages of the current block are marked invalid, then the boot block index is incremented by 1, and FCM repeats step 3. eLBC will continue searching for a bootable block indefinitely, therefore at least one block must be marked valid for boot loading to proceed. At the conclusion of the boot block search, the value of $\text{FBAR}[\text{BLK}]$ points to the boot block.

4. If ECC checking is enabled, the FCM recovers from the spare region the stored ECC for each 512-byte block of boot data. The boot block must be prepared with ECC protection. During ECC generation, software should use $\text{FMR}[\text{ECCM}] = 0$ for small-page devices, and $\text{FMR}[\text{ECCM}] = 1$ for large-page devices.

If RCW initialization is required, the first 64 bytes of the boot block must be prepared in accordance with the layout described in .”

5. FCM performs a sequence of random-access page reads, reading entire pages from the boot block until 4 Kbytes have been saved to the FCM buffer RAM. If ECC checking is enabled, the ECC of each 512-byte region is verified and single-bit errors are corrected if possible. If FCM is unable to correct ECC errors, eLBC halts the boot process and signals an unrecoverable error by asserting the $\overline{\text{hreset_req}}$ signal.

- The CPU now commences fetching instructions, in random order, from the FCM buffer RAM. This first-level boot loader typically copies a secondary boot loader into system memory, and continues booting from there. Boot software must clear FMR[BOOT] to enable normal operation of FCM.

11.4.4 User-Programmable Machines (UPMs)

UPMs are flexible interfaces that connect to a wide range of memory devices. At the heart of each UPM is an internal RAM array that specifies the logical value driven on the external memory control signals (\overline{LCSn} , $\overline{LBS}[0:1]$ and $\overline{LGPL}[0:5]$) for a given clock cycle. Each word in the RAM array provides bits that allow a memory access to be controlled with a resolution of up to one quarter of the external bus clock period on the byte-select and chip-select lines. A gap of 2 dead LCLK cycles is present on the UPM interface between UPM transactions.

NOTE

If the $\overline{LGPL4}/\overline{LGTA}/\overline{LFRB}/\overline{LUPWAIT}$ signal is used as both an input and an output, a weak pull-up is required. Refer to the *MPC8309 PowerQUICC II Pro Integrated Communications Processor Hardware Specification* for details regarding termination options.

Figure 11-56 shows the basic operation of each UPM.

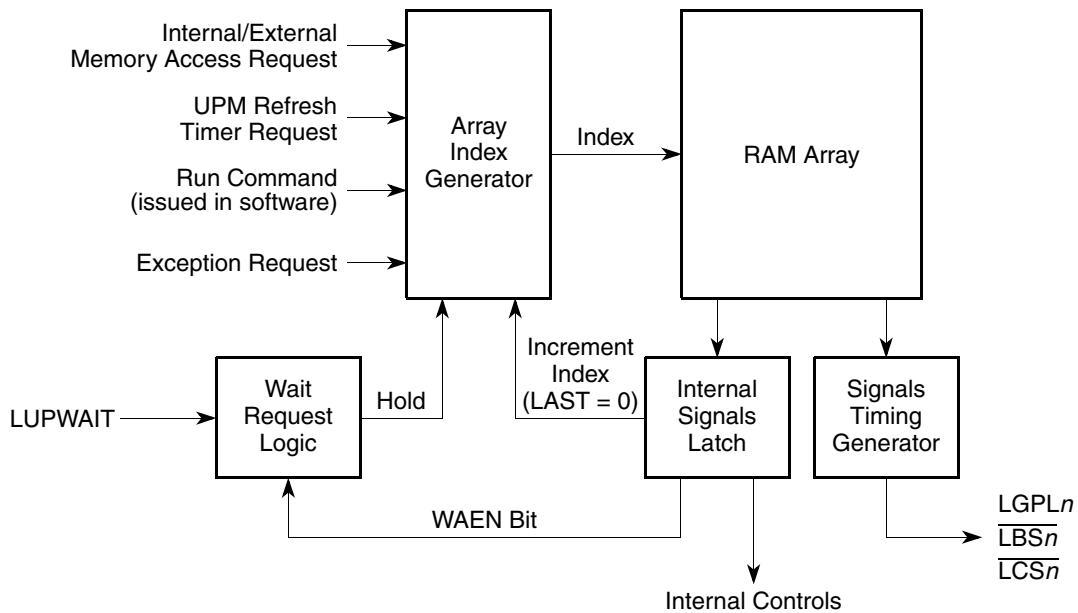


Figure 11-56. User-Programmable Machine Functional Block Diagram

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.

11.4.4.1 UPM Requests

A special pattern location in the RAM array is associated with each of the possible UPM requests. An internal device's request for a memory access initiates one of the following patterns (MxMR[OP] = 00):

- Read single-beat pattern (RSS)
- Read burst cycle pattern (RBS)
- Write single-beat pattern (WSS)
- Write burst cycle pattern (WBS)

A UPM refresh timer request pattern initiates a refresh timer pattern (RTS).

An exception (caused by a bus monitor time-out error) occurring while another UPM pattern is running initiates an exception condition pattern (EXS).

Figure 11-57 and Table 11-37 show the start addresses of these patterns in the UPM RAM, according to cycle type. RUN commands (MxMR[OP] = 11), however, can initiate patterns starting at any of the 64 UPM RAM words.

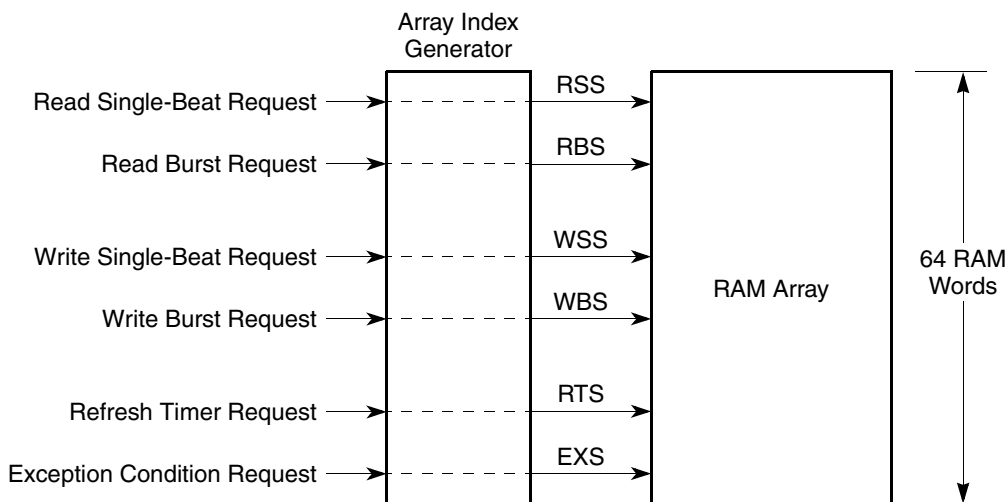


Figure 11-57. RAM Array Indexing

Table 11-37. UPM Routines Start Addresses

UPM Routine	Routine Start Address
Read single-beat (RSS)	0x00
Read burst (RBS)	0x08
Write single-beat (WSS)	0x18
Write burst (WBS)	0x20

Table 11-37. UPM Routines Start Addresses (continued)

UPM Routine	Routine Start Address
Refresh timer (RTS)	0x30
Exception condition (EXS)	0x3C

11.4.4.1.1 Memory Access Requests

The user must ensure that the UPM is appropriately initialized before a request occurs.

The UPM supports two types of memory reads and writes:

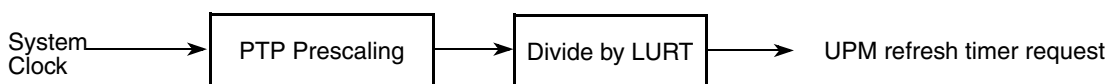
- A single-beat transfer transfers one operand consisting of up to a single word (dependent on port size). A single-beat cycle starts with one transfer start and ends with one transfer acknowledge.
- A burst transfer transfers exactly 4 double words regardless of port size. For 32-bit accesses, the burst cycle starts with one transfer start but ends after eight transfer acknowledges, whereas an 8-bit device requires 32 transfer acknowledges.

The user must ensure that patterns for single-beat transfers contain one, and only one, transfer acknowledge (UTA bit in RAM word set high) and for a burst transfer, contain the exact number of transfer acknowledges required.

Any transfers that do not naturally fit single or burst transfers are synthesized as a series of single transfers. These accesses are treated by the UPM as back-to-back, single-beat transfers. Burst transfers can also be inhibited by setting $OR_n[BI]$. Burst performance can be achieved by ensuring that UPM transactions are 32-byte aligned with a transaction size being some multiple of 32-bytes, which is a natural fit for cache-line transfers, for example.

11.4.4.1.2 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 11-58](#) 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 11-58. Memory Refresh Timer Request Block Diagram

By default, all local bus refreshes are performed using the refresh pattern of UPMA. This means that if refresh is required, MAMR[RFEN] must be set. It also means that only one refresh routine should be programmed and be placed in UPMA, which serves as the refresh executor. Any banks assigned to a UPM are provided with the common UPMA refresh pattern if the RFEN bit of the corresponding UPM is set, concurrently. UPMA assigned banks, therefore, always receive refresh services when MAMR[RFEN] is set, while UPMB and UPMC assigned banks also receive (the same) refresh services if the corresponding MxMR[RFEN] bits are set. In this scenario, more than one chip select may assert at the same time, as refresh pattern runs for all banks assigned to UPM with RFEN bit set.

11.4.4.1.3 Software Requests—RUN Command

Software can start a request to the UPM by issuing a RUN command to the UPM. Some memory devices have their own signal handshaking protocol to put them into special modes, such as self-refresh mode.

For these special cycles, the user creates a special RAM pattern that can be stored in any unused areas in the UPM RAM. Then a RUN command is used to run the cycle. The UPM runs the pattern beginning at the specified RAM location until it encounters a RAM word with its LAST bit set. The RUN command is issued by setting $MxMR[OP] = 11$ and accessing UPM_n memory region with any write transaction that hits the corresponding UPM machine. $MxMR[MAD]$ 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.

11.4.4.1.4 Exception Requests

When the eLBC under UPM control initiates an access to a memory device and an exception occurs (bus monitor time-out), the UPM provides a mechanism by which memory control signals can meet the device's timing requirements without losing data. The mechanism is the exception pattern that defines how the UPM negates its signals in a controlled manner.

11.4.4.2 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 $MxMR$.

Patterns are written to the RAM array by setting $MxMR[OP] = 01$ 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 $MxMR[OP] = 10$).

NOTE

$MxMR$ /MDR registers should not be updated while dummy read/write access is still in progress. If the $MxMR[MAD]$ is incremented then the previous dummy transaction is already completed.

In order to enforce proper ordering between updates to the MxMR/MDR register and the dummy accesses to the UPM memory region, two rules must be followed:

- Since the result of any update to the MxMR/MDR register must be in effect before the dummy read or write to the UPM region, a write to MxMR/MDR should be followed immediately by a read of MxMR/MDR.
- The UPM memory region should have the same MMU settings as the memory region containing the MxMR configuration register; both should be mapped by the MMU as cache-inhibited and guarded. This prevents the CPU from re-ordering a read of the UPM memory around the read of MxMR. Once the programming of the UPM array is complete the MMU setting for the associated address range can be set to the proper mode for normal operation, such as cacheable and copyback.

For proper signalling, the following guidelines must be followed while programming UPM RAM words:

- For UPM reads, program UTA and LAST in the same or consecutive RAM words.
- For UPM burst reads, program last UTA and LAST in the same or consecutive RAM words.
- For UPM writes, program UTA and LAST in the same RAM word.
- For UPM burst writes, program last UTA and LAST in the same RAM word.

11.4.4.2.1 UPM Programming Example (Two Sequential Writes to the RAM Array)

The following example further illustrates the steps required to perform two writes to the RAM array at non-sequential addresses assuming that the relevant BR n and OR n registers have been previously set up:

1. Program MxMR for the first write (with the desired RAM array address).
2. Write pattern/data to MDR to ensure that the MxMR has already been updated with the desired configuration.
3. Read MDR to ensure that the MDR has already been updated with the desired pattern. (Or, read MxMR register if step 2 is not performed.)
4. Perform a dummy write transaction.
5. Read/check MxMR[MAD]. If incremented, the previous dummy write transaction is completed; proceed to step 6. Repeat step 5 until incremented.
6. Program MxMR for the second write with the desired RAM array address.
7. Write pattern/data to MDR to ensure that the MxMR has already been updated with the desired configuration.
8. Read MDR to ensure that the MDR has already been updated with the desired pattern.
9. Perform a dummy write transaction.
10. Read/check MxMR[MAD]. If incremented, the previous dummy write transaction is completed.

Note that if steps 1 and 2 or steps 6 and 7 are reversed, step 3 or 8 (as appropriate) is replaced by the following:

- Read MxMR to ensure that the MxMR has already been updated with the desired configuration.

11.4.4.2 UPM Programming Example (Two Sequential Reads from the RAM Array)

RAM array contents may also be read for debug purposes, for example, by alternating dummy read transactions, each time followed by reads of MDR (MxMR[OP] = 0b10). The following example further illustrates the steps required to perform two reads from the RAM array at non-sequential addresses assuming that the relevant BR_n and OR_n registers have been previously set up:

1. Program MxMR for the first read with the desired RAM array address.
2. Read MxMR to ensure that the MxMR has already been updated with the desired configuration, such as RAM array address.
3. Perform a dummy read transaction.
4. Read/check MxMR[MAD]. If incremented, the previous dummy read transaction is completed; proceed to step 5. Repeat step 4 until incremented.
5. Read MDR.
6. Program MxMR for the second read with the desired RAM array address.
7. Read MxMR to ensure that the MxMR has already been updated with the desired configuration, such as RAM array address.
8. Perform a dummy read transaction.
9. Read/check MxMR[MAD]. If incremented, the previous dummy read transaction is completed; proceed to step 10. Repeat step 9 until incremented.
10. Read MDR.

11.4.4.3 UPM Signal Timing

RAM word fields specify the value of the various external signals at a granularity of up to four values for each bus clock cycle. The signal timing generator causes external signals to behave according to timing specified in the current RAM word. For LCRR[CLKDIV] = 4 or 8, each bit in the RAM word relating to \overline{LCS}_n and \overline{LBS} 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 11-59](#) and [Figure 11-60](#). 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.

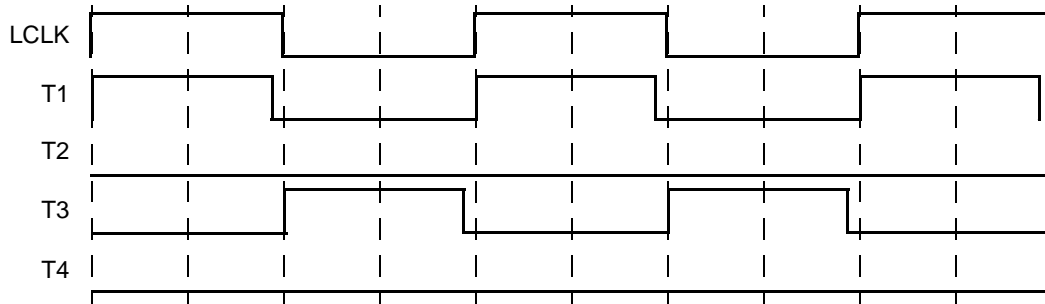


Figure 11-59. UPM Clock Scheme for LCRR[CLKDIV] = 2

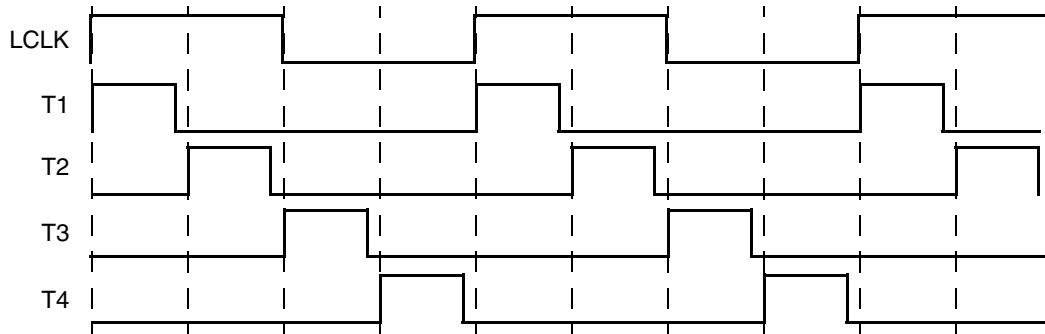


Figure 11-60. UPM Clock Scheme for LCRR[CLKDIV] = 4 or 8

11.4.4.4 RAM Array

The RAM array for each UPM is 64 locations deep and 32 bits wide, as shown in Figure 11-61. The signals at the bottom of the figure are UPM outputs. The selected \overline{LCS}_n is for the bank that matches the current address. The selected \overline{LBS} is for the byte lanes read or written by the access.

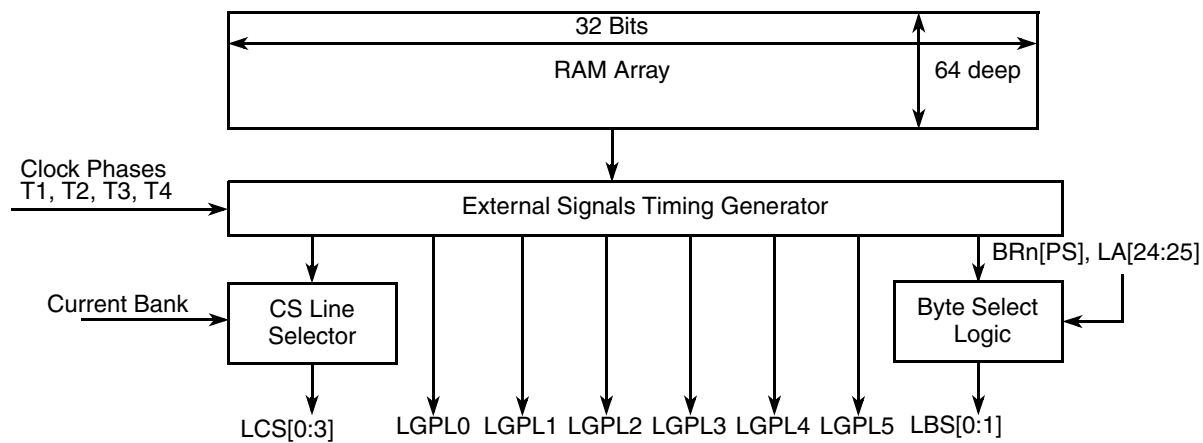


Figure 11-61. RAM Array and Signal Generation

11.4.4.4.1 RAM Words

The RAM word is a 32-bit microinstruction stored in one of 64 locations in the RAM array. It specifies timing for external signals controlled by the UPM. Figure 11-35 shows the RAM word fields. When $LCRR[CLKDIV] = 4$ or 8 , the CST_n and BST_n bits determine the state of UPM signals \overline{LCS}_n and $\overline{LBS}[0:1]$ at each quarter phase of the bus clock. When $LCRR[CLKDIV] = 2$, CST_2 and CST_4 are ignored and the external has the values defined by CST_1 and CST_3 but extended to half the clock cycle in duration. The same interpretation occurs for the BST_n bits when $LCRR[CLKDIV] = 2$.

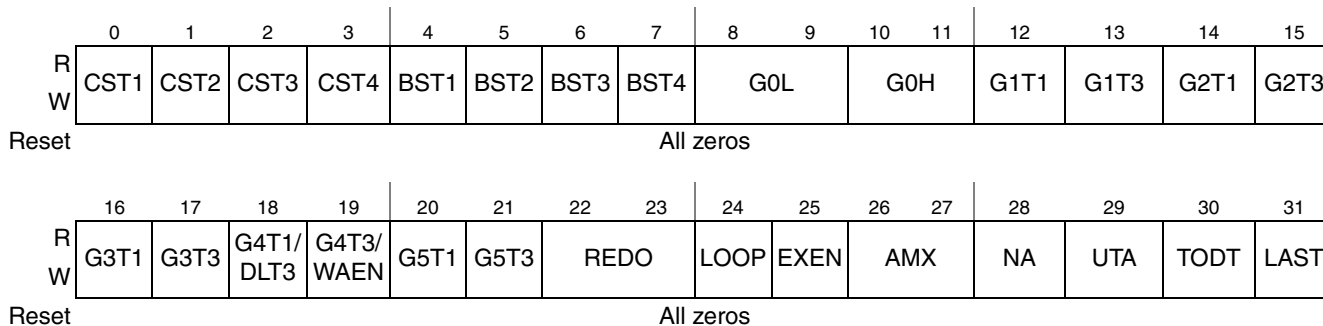


Figure 11-62. RAM Word Fields

Table 11-38 contains descriptions of the RAM word fields.

Table 11-38. RAM Word Field Descriptions

Bits	Name	Description
0	CST1	Chip select timing 1. Defines the state (0 or 1) of \overline{LCS}_n during bus clock quarter phase 1 if $LCRR[CLKDIV] = 4$ or 8 . Defines the state (0 or 1) of \overline{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{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{LCS}_n during bus clock quarter phase 3 if $LCRR[CLKDIV] = 4$ or 8 . Defines the state (0 or 1) of \overline{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{LCS}_n during bus clock quarter phase 4 if $LCRR[CLKDIV] = 4$ or 8 . Ignored when $LCRR[CLKDIV] = 2$.
4	BST1	Byte select timing 1. Defines the state (0 or 1) of \overline{LBS} during bus clock quarter phase 1 ($LCRR[CLKDIV] = 4$ or 8) or bus clock half phase 1 ($LCRR[CLKDIV] = 2$), in conjunction with $BR_n[PS]$ and $LA[24:25]$.
5	BST2	Byte select timing 2. Defines the state (0 or 1) of \overline{LBS} during bus clock quarter phase 2 ($LCRR[CLKDIV] = 4$ or 8), in conjunction with $BR_n[PS]$ and $LA[24:25]$. Ignored when $LCRR[CLKDIV] = 2$.
6	BST3	Byte select timing 3. Defines the state (0 or 1) of \overline{LBS} during bus clock quarter phase 3 ($LCRR[CLKDIV] = 4$ or 8) or bus clock half phase 2 ($LCRR[CLKDIV] = 2$), in conjunction with $BR_n[PS]$ and $LA[24:25]$.
7	BST4	Byte select timing 4. Defines the state (0 or 1) of \overline{LBS} during bus clock quarter phase 4 ($LCRR[CLKDIV] = 4$ or 8), in conjunction with $BR_n[PS]$ and $LA[24:25]$. Ignored when $LCRR[CLKDIV] = 2$.

Table 11-38. 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 MxMR[G0CL] 01 Reserved 10 0 11 1
10–11	G0H	General purpose line 0 higher. Defines the state of LGPL0 during the bus clock quarter phases 3 and 4 (second half phase). 00 Value defined by MxMR[G0CL] 01 Reserved 10 0 11 1
12	G1T1	General purpose line 1 timing 1. Defines the state (0 or 1) of LGPL1 during bus clock quarter phases 1 and 2 (first half phase).
13	G1T3	General purpose line 1 timing 3. Defines the state (0 or 1) of LGPL1 during bus clock quarter phases 3 and 4 (second half phase)
14	G2T1	General purpose line 2 timing 1. Defines state (0 or 1) of LGPL2 during bus clock quarter phases 1 and 2 (first half phase).
15	G2T3	General purpose line 2 timing 3. Defines the state (0 or 1) of LGPL2 during bus clock quarter phases 3 and 4 (second half phase).
16	G3T1	General purpose line 3 timing 1. Defines the state (0 or 1) of LGPL3 during bus clock quarter phases 1 and 2 (first half phase).
17	G3T3	General purpose line 3 timing 3. Defines the state (0 or 1) of LGPL3 during bus clock quarter phases 3 and 4 (second half phase).
18	G4T1/DLT3	General purpose line 4 timing 1/delay time 3. The function of this bit is determined by MxMR[GPL4]. If MxMR[GPL4] = 0 and LGPL4/LUPWAIT pin functions as an output (LGPL4), G4T1/DLT3 defines the state (0 or 1) of LGPL4 during bus clock quarter phases 1 and 2 (first half phase). If MxMR[GPL4] = 1 and LGPL4/LUPWAIT functions as an input (LUPWAIT), if a read burst or single read is executed, G4T1/DLT3 defines the sampling of the data bus as follows: 0 In the current word, the data bus should be sampled at the start of bus clock quarter phase 1 of the next bus clock cycle. 1 In the current word, the data bus should be sampled at the start of bus clock quarter phase 3 of the current bus clock cycle.
19	G4T3/WAEN	General purpose line 4 timing 3/wait enable. Bit function is determined by MxMR[GPL4]. If MxMR[GPL4] = 0 and LGPL4/LUPWAIT pin functions as an output (LGPL4), G4T3/WAEN defines the state (0 or 1) of LGPL4 during bus clock quarter phases 3 and 4 (second half phase). If MxMR[GPL4] = 1 and LGPL4/LUPWAIT functions as an input (LUPWAIT), G4T3/WAEN is used to enable the wait mechanism: 0 LUPWAIT detection is disabled. 1 LUPWAIT is enabled. If LUPWAIT is detected as being asserted, a freeze in the external signals logical values occurs until LUPWAIT is detected as being negated.
20	G5T1	General purpose line 5 timing 1. Defines the state (0 or 1) of LGPL5 during bus clock quarter phases 1 and 2 (first half phase).
21	G5T3	General purpose line 5 timing 3. Defines the state (0 or 1) of LGPL5 during bus clock quarter phases 3 and 4 (second half phase).

Table 11-38. RAM Word Field Descriptions (continued)

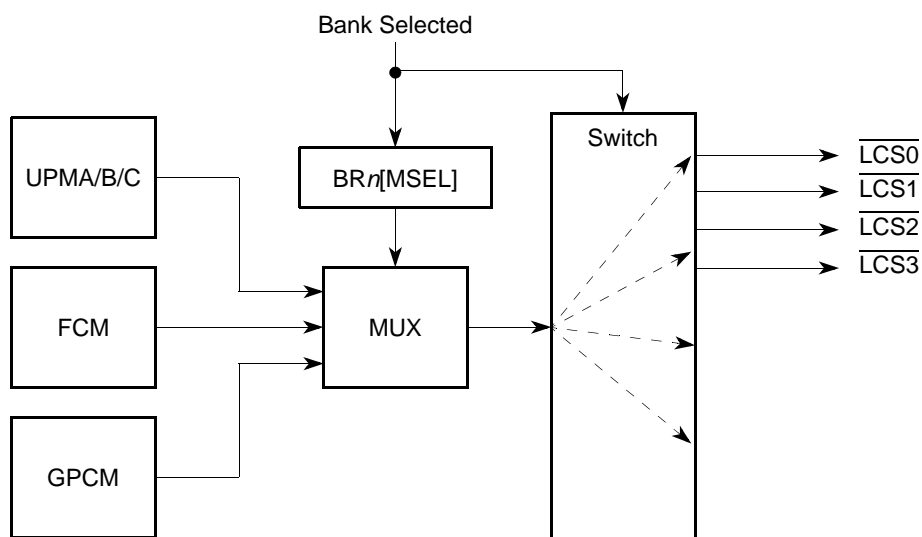
Bits	Name	Description
22–23	REDO	Redo current RAM word. Defines the number of times to execute the current RAM word. 00 Once (normal operation) 01 Twice 10 Three times 11 Four times
24	LOOP	Loop start/end. The first RAM word in the RAM array where LOOP is 1 is recognized as the loop start word. The next 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 MxMR. 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. Note: AMX must not change values in any RAM word which begins 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 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 during an LALE phase. Any change in the AMX field initiates a new LALE (address) phase. 00 LAD (and/or in conjunction with LA) is the non-multiplexed address. For example, column address. 01 Reserved 10 LAD (and/or in conjunction with LA) is driven with the multiplexed address according to MxMR[AM]. For example, row address. See Section 11.4.4.4.7, “Address Multiplexing (AMX)” for more information. 11 LAD (and/or in conjunction with LA) is driven with the contents of MAR. Used, for example, to initialize a mode. Note: Source ID debug mode is only supported for the AMX = 00 setting. Note: AMX must not change values in any RAM word which begins a loop.
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 BRn[PS], the increment value of LAN is 1 or 2 for port sizes of 8 and 16 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. In case of UPM writes, program UTA and LAST in same RAM word. In case of UPM reads, program UTA and LAST in consecutive or same RAM words.

Table 11-38. RAM Word Field Descriptions (continued)

Bits	Name	Description
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 $MxMR[DSn]$. The disable timer does not affect memory accesses to different banks. Note that TODT must be set together with LAST, otherwise it is ignored. 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.
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 $MxMR[DSn]$. 0 The UPM continues executing RAM words. 1 Indicates the last RAM word in the program. The service to the UPM request is done after this cycle concludes. In case of UPM writes, program UTA and LAST in same RAM word. In case of UPM reads, program UTA and LAST in consecutive or same RAM words.

11.4.4.4.2 Chip-Select Signal Timing ($CSTn$)

If $BRn[MSEL]$ of the accessed bank selects a UPM on the currently requested cycle, the UPM manipulates the \overline{LCSn} for that bank with timing as specified in the UPM RAM word $CSTn$ fields. The selected UPM affects only the assertion and negation of the appropriate \overline{LCSn} signal. The state of the selected \overline{LCSn} signal of the corresponding bank depends on the value of each $CSTn$ bit. Figure 11-63 shows how UPMs control \overline{LCSn} signals.


Figure 11-63. \overline{LCSn} Signal Selection

11.4.4.4.3 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 $\overline{\text{LBS}}[0:1]$ signal. The timing of both byte-select signals is specified in the RAM word. However, $\overline{\text{LBS}}[0:1]$ are also controlled by the port size of the accessed bank, the number of bytes to transfer, and the address accessed. Figure 11-64 shows how UPMs control $\overline{\text{LBS}}[0:1]$.

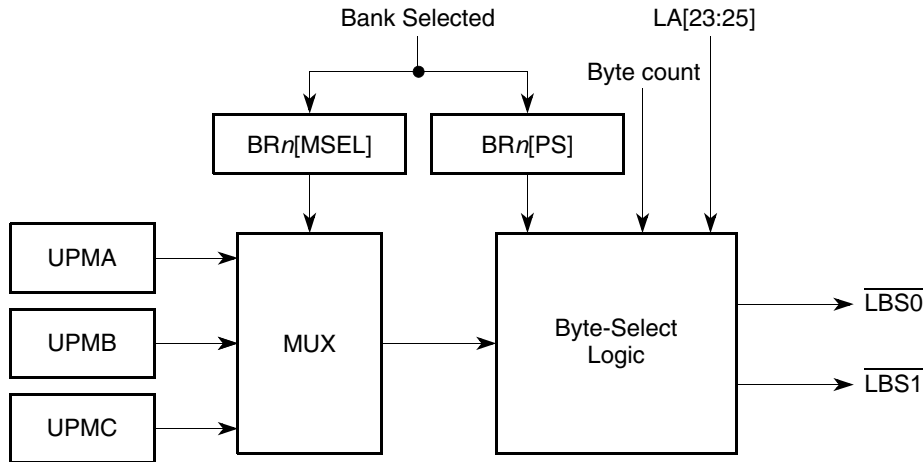


Figure 11-64. $\overline{\text{LBS}}$ Signal Selection

The uppermost byte select ($\overline{\text{LBS}}_0$), when asserted, indicates that LAD[0:7] contains valid data during a cycle. Likewise, $\overline{\text{LBS}}_1$ indicates that LAD[8:15] contain valid data. For a UPM refresh timer request, all $\overline{\text{LBS}}[0:1]$ signals are asserted/negated by the UPM according to the refresh pattern only. Following any internal bus monitor exception, the $\overline{\text{LBS}}[0:1]$ signals are negated regardless of the exception handling provided by any UPM exception pattern to prevent spurious writes to external RAM.

11.4.4.4.4 General-Purpose Signals (GnT_n, GOn)

The general-purpose signals (LGPL[0:5]) each have two bits in the RAM word that define the logical value of the signal to be changed at the rising edge of the bus clock and/or at the falling edge of the bus clock. LGPL0 offers enhancements beyond the other LGPL_n lines.

LGPL0 can be controlled by an address line specified in MxMR[G0CL]. To use this feature, G0H and G0L should be set in the RAM word. For example, for a SIMM with multiple banks, this address line can be used to switch between internal memory device banks.

11.4.4.4.5 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 11-39. The next RAM word for which LOOP = 1 is recognized as a loop end word. When it is reached, the loop counter is decremented by one.

Continued loop execution depends on the loop counter. If the counter is not zero, the next RAM word executed is the loop start word. Otherwise, the next RAM word executed is the one after the loop end word. Loops can be executed sequentially but cannot be nested. Also, special care must be taken:

- LAST and LOOP must not be set together.
- Loop start word should not have an AMX change with regard to the previous word.

Table 11-39. MxMR Loop Field Use

Request Serviced	Loop Field
Read single-beat cycle	RLF
Read burst cycle	RLF
Write single-beat cycle	WLF
Write burst cycle	WLF
Refresh timer expired	TLF
RUN command	RLF

11.4.4.4.6 Repeat Execution of Current RAM Word (REDO)

The REDO function is useful for wait-state insertion in a long UPM routine that would otherwise need too many RAM words. Setting the REDO bits of the RAM word to a nonzero value causes the UPM to re-execute the current RAM word up to three more times, as defined in the REDO field of the current RAM word.

Special care must be taken in the following cases:

- When UTA and REDO are set together, TA is asserted the number of times specified by the REDO function.
- When NA and REDO are set together, the address is incremented the number of times specified by the REDO function.
- When LOOP and REDO are set together, the loop mechanism works as usual and the line is repeated according to the REDO function.
- LAST and REDO must not be set together.
- REDO should not be used within the exception routine.

11.4.4.4.7 Address Multiplexing (AMX)

Address lines can be controlled by the user-provided pattern in the UPM. The address multiplex (AMX) bits in the RAM word can choose between driving the transaction address (AMX = 00), driving it according to the multiplexing specified by the MxMR[AM] field (AMX = 10), or driving the contents of MAR (AMX = 11) on the address signals. The next address (NA) bit of the RAM word does not affect LA signals, unless AMX = 00 and chooses the column address for NA = 1.

In all cases, LA[21:25] of the eLBC are driven by the five lsbs of the address selected by AMX, regardless of whether the next address (NA) bit of the RAM word is used to increment the current address. The effect of NA = 1 is visible only when AMX = 00 chooses the column address.

Table 11-40 shows how the RAM word AMX bits and MxMR[AM] settings can be used to affect row × column address multiplexing on the [10:25] signals.

NOTE

Multiple-bank DRAM and SDRAM devices require that the bank address be driven during both RAS and CAS cycles. The UPM does not support a persistent bank address on both RAS and CAS cycles. Therefore, external logic must be used to supply a bank address to these devices.

Table 11-40. UPM Address Multiplexing

	Internal Transaction Address																																									
	msb	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	lsb								
AMX = 10 MxMR[AM] = 000 (Row)										LAD						LA																										
											10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25																
AMX = 00 (Col)																											LA															
AMX = 10 MxMR[AM] = 010 (Row)										LAD						LA																										
											10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25																
AMX = 00 (Col)																																										
AMX = 10 MxMR[AM] = 100 (Row)										LAD						LA																										
											10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25																
AMX = 00 (Col)																																										
AMX = 10 MxMR[AM] = 110 (Row)										LAD						LA																										
											10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25																
AMX = 00 (Col)																																										
AMX = 10 MxMR[AM] = 111 (Row)										Reserved																																
AMX = 00 (Col)										Reserved																																

Note that any change to the AMX field from one RAM word to the next RAM word executed results in an address phase on the {LADn, LAN} bus with the assertion of LALE for the number of cycles set for LALE in the ORn and LCRR registers. The LGPL[0:5] signals maintain the value specified in the RAM word during the LALE phase.

NOTE

AMX must not change values in any RAM word which begins a loop.

11.4.4.4.8 Data Valid and Data Sample Control (UTA)

When a read access is handled by the UPM, and the UTA bit is 1 (data is to be sampled by the eLBC), the value of the DLT3 bit in the same RAM word, in conjunction with $MxMR[GPL4]$, determines when the data input is sampled by the eLBC as follows:

- If $MxMR[GPL4] = 1$ (G4T4/DLT3 functions as DLT3) and $DLT3 = 1$ in the RAM word, data is latched on the falling edge of the bus clock instead of the rising edge. The eLBC samples the data on the next falling edge of the bus clock, which is during the middle of the current bus cycle. This feature should be used only in systems without external synchronous bus devices that require mid-cycle sampling.
- If $MxMR[GPL4] = 0$ (G4T4/DLT3 functions as G4T4), or if $MxMR[GPL4] = 1$ but $DLT3 = 0$ in the RAM word, data is latched on the rising edge of the bus clock, which occurs at the end of the current bus clock cycle (normal operation).

Figure 11-65 shows how data sampling is controlled by the UPM.

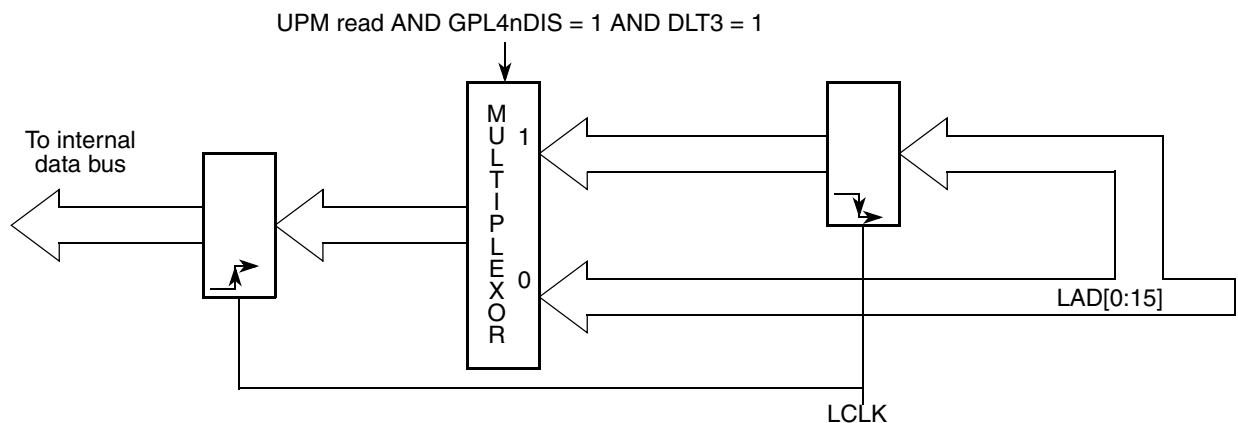


Figure 11-65. UPM Read Access Data Sampling

11.4.4.4.9 LGPL[0:5] Signal Negation (LAST)

When the LAST bit is read in a RAM word, the current UPM pattern is terminated at the end of the current cycle. On the next cycle (following LAST) all the UPM signals are negated unconditionally (driven to logic 1), unless there is a back-to-back UPM request pending. In this case, the signal values for the cycle following the one in which the LAST bit was set are taken from the first RAM word of the pending UPM routine.

In case of UPM writes, program UTA and LAST in same RAM word. In case of UPM reads, program UTA and LAST in consecutive or same RAM words.

11.4.4.4.10 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 11-66 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{LCSn} and LGPL1 states and the WAEN value are frozen until LUPWAIT is recognized as negated. WAEN is typically set before the line that contains UTA = 1. Note that if WAEN and NA are both set in the same RAM word, NA causes the burst address to increment once as normal regardless of whether the UPM freezes.

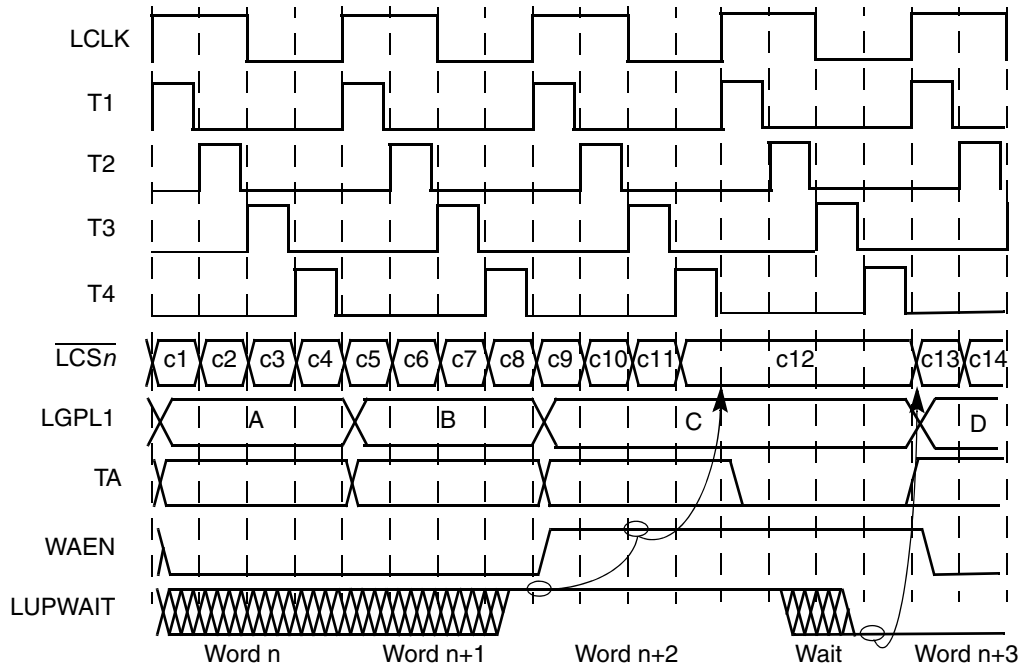


Figure 11-66. Effect of LUPWAIT Signal

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.

11.4.4.5 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 $ORn[TRLX]$ and $ORn[EHTR]$. The next accesses after a read access to the slow memory device is delayed by the number of clock cycles specified in the ORn register in addition to any existing bus turnaround cycle.

11.5 Initialization/Application Information

This section provides information about the following:

- [Section 11.5.1, “Interfacing to Peripherals in Different Address Modes”](#)
- [Section 11.5.2, “Bus Turnaround”](#)
- [Section 11.5.3, “Interface to Different Port-Size Devices”](#)
- [Section 11.5.4, “Command Sequence Examples for NAND Flash EEPROM”](#)
- [Section 11.5.5, “Interfacing to Fast-Page Mode DRAM Using UPM”](#)
- [Section 11.5.6, “Interfacing to ZBT SRAM Using UPM”](#)

11.5.1 Interfacing to Peripherals in Different Address Modes

This section provides guidelines for interfacing to peripherals.

11.5.1.1 Multiplexed Address/Data Bus for 26-Bit Addressing

In this mode, the eLBC is used with port sizes of 8 and 16 bits; address bits A[6:21] will appear on LAD[0:15] during address phases, while the lower 10 bits of the address, A[22:31], are driven permanently on LA[16:25]. The connection is illustrated in [Figure 11-67](#).

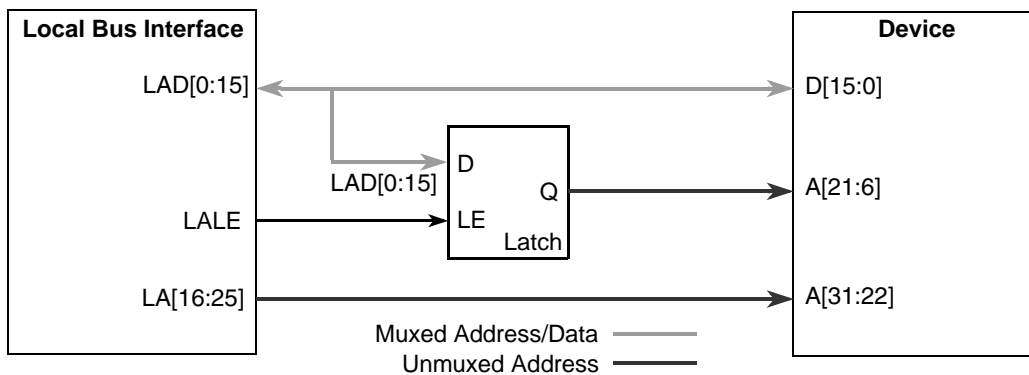


Figure 11-67. Multiplexed Address/Data Bus for 26-Bit Addressing

11.5.1.2 Peripheral Hierarchy on the Local Bus for High Bus Speeds

To achieve the highest possible bus speeds on the local bus, it is recommended to reduce the number of devices connected directly to the bus. For best results, only one bank of synchronous SRAMs should have a direct connection, and a bus demultiplexer should be used to replace separate latch and separate bus transceiver combinations. [Figure 11-68](#) shows an example of such a hierarchy. This section is only a

guideline, and the board designer must simulate the electric characteristics of the scenario to determine the maximum operating frequency.

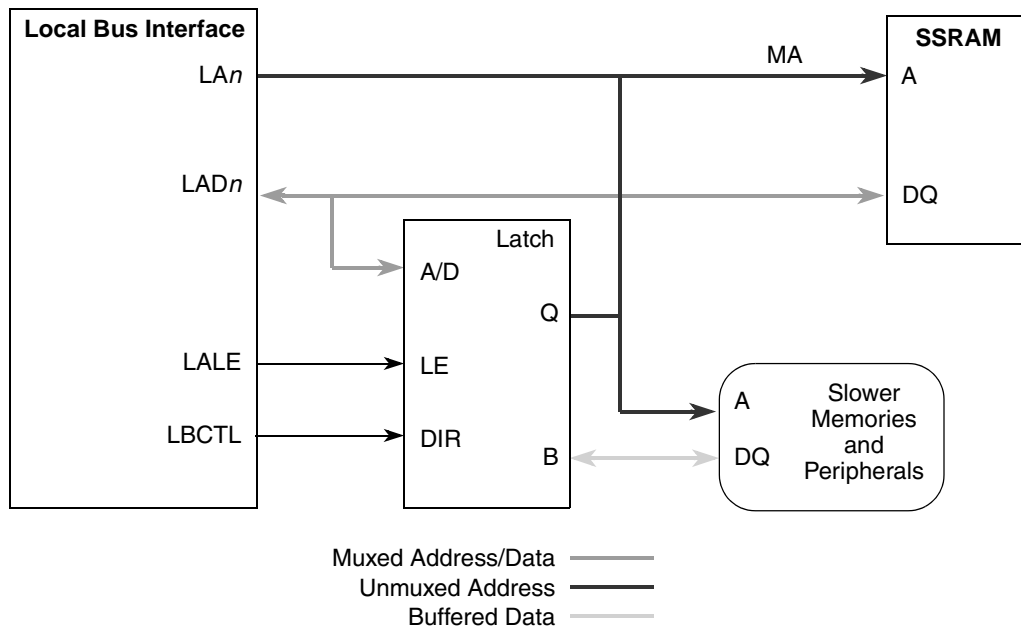


Figure 11-68. Local Bus Peripheral Hierarchy for High Bus Speeds

11.5.1.3 GPCM Timings

In case a system contains a memory hierarchy with high speed synchronous memories (synchronous SRAM) and lower speed asynchronous memories (for example, FLASH EPROM and peripherals) the GPCM-controlled memories should be decoupled by buffers to reduce capacitive loading on the bus. Those buffers have to be taken into account for the timing calculations. Figure 11-69 illustrates GPCM address timings.

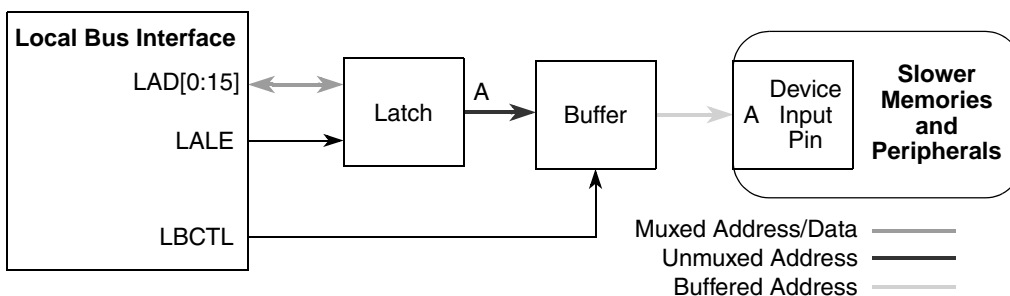


Figure 11-69. GPCM Address Timings

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 to 6 ns, so for a 100-MHz bus frequency, \overline{LCS} should arrive on the order of 2 to 3 bus clocks later.

For data timings, only the propagation delay of one buffer plus the actual data setup time has to be considered. See [Figure 11-70](#) for an illustration of GPCM data timings.

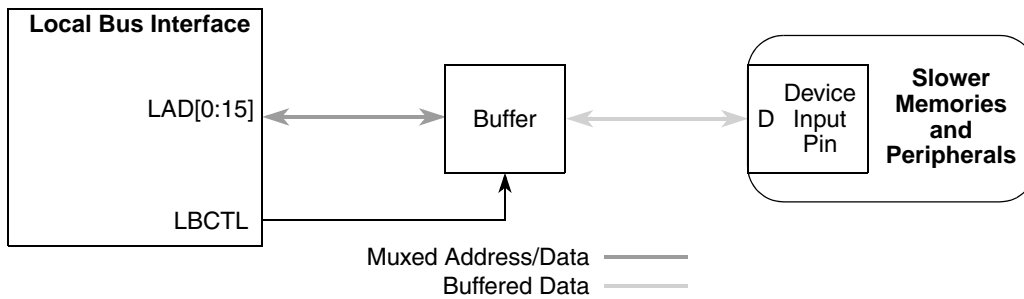


Figure 11-70. GPCM Data Timings

11.5.2 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

11.5.2.1 Address Phase after Previous Read

During a read cycle, the memory/peripheral drives the bus and the bus transceiver drives LAD. After the data has been sampled, the output drivers of the external device must be disabled. This can take some time; for slow devices the EHTR feature of the GPCM or the programmability of the UPM should be used to guarantee that those devices have stopped driving the bus when the eLBC memory controller ends the bus cycle.

In this case, after the previous cycle ends, LBCTL goes high and changes the direction of the bus transceiver. The eLBC then inserts a bus turnaround cycle to avoid contention. The external device has now already placed its data signals in high impedance and no bus contention will occur.

11.5.2.2 Read Data Phase after Address Phase

During the address phase, LAD actively drives the address and LBCTL is high, driving the bus transceivers in the same direction as during a write. After the end of the address phase, LBCTL goes low and changes the direction of the bus transceiver. The eLBC places the LAD signals in high impedance after its $t_{dis}(LB)$. The LBCTL will have its new state after $t_{en}(LB)$ and, because this is an asynchronous input,

the transceiver starts to drive those signals after its t_{en} (transceiver) time. The system designer has to ensure, that $[t_{en}(LB) + t_{en}(\text{transceiver})]$ is larger than $t_{dis}(LB)$ to avoid bus contention.

11.5.2.3 Read-Modify-Write Cycle for Parity Protected Memory Banks

Principally, a read-modify-write cycle is a read cycle immediately followed by a write cycle. Because the write cycle will have a new address phase in any case, this basically is the same case as an address phase after a previous read.

11.5.2.4 UPM Cycles with Additional Address Phases

The flexibility of the UPM allows the user to insert additional address phases during read cycles by changing the AMX field, therefore turning around the bus during one pattern. The eLBC automatically inserts a single bus turnaround cycle if the bus (LAD) was previously high impedance for any reason, such as a read, before LALE is driven and LAD is driven with the new address. The turnaround cycle is not inserted on a write, because the bus was already driven to begin with.

However, bus contention could potentially still occur on the far side of a bus transceiver. It is the responsibility of the designer of the UPM pattern to guarantee that enough idle cycles are inserted in the UPM pattern to avoid this.

11.5.3 Interface to Different Port-Size Devices

The eLBC supports 8- and 16-bit data port sizes. However, the bus requires that the portion of the data bus used for a transfer to or from a particular port size be fixed. A 16-bit port must reside on LAD[0–15], and an 8-bit port must reside on LAD[0–7]. The local bus always tries to transfer the maximum amount of data on all bus cycles. [Figure 11-71](#) shows the device connections on the data bus.

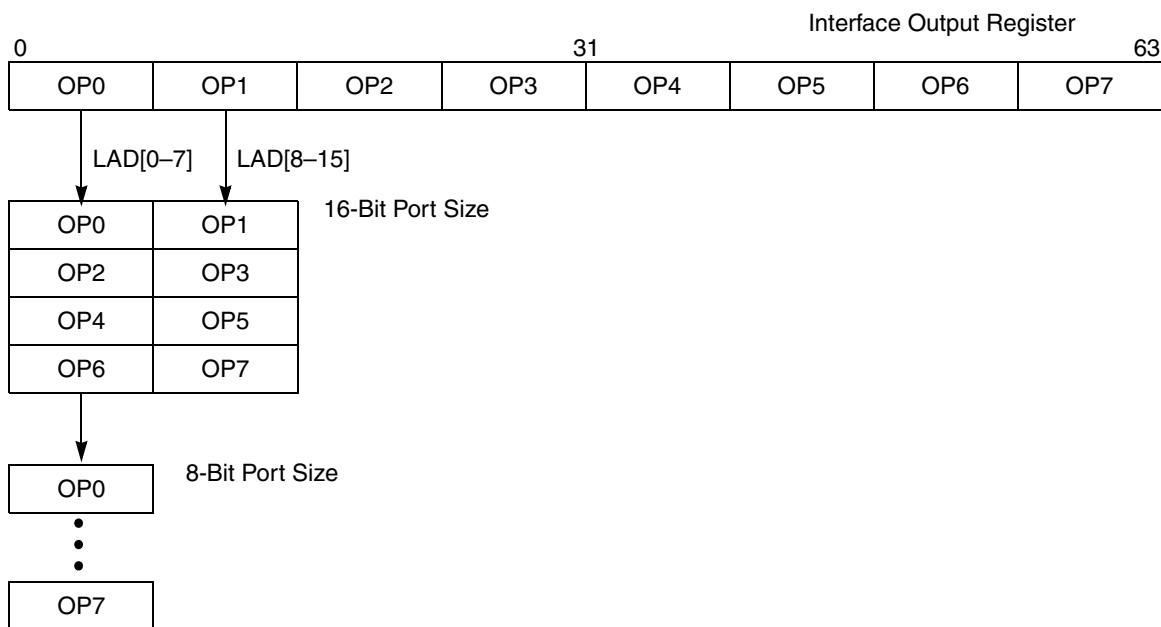


Figure 11-71. Interface to Different Port-Size Devices

Table 11-41 lists the bytes required on the data bus for read cycles.

Table 11-41. Data Bus Drive Requirements For Read Cycles

Transfer Size	Address State ¹ 3 lsbs	Port Size/LAD Data Bus Assignments							
		16-Bit				8-Bit			
		0-7	8-15	16-23	24-31	0-7	8-15	16-23	24-31
Byte	000	OP0	—			OP0			
	001	—	OP1			OP1			
	010	OP2	—			OP2			
	011	—	OP3			OP3			
	100	OP4	—			OP4			
	101	—	OP5			OP5			
	110	OP6	—			OP6			
	111	—	OP7			OP7			
Half Word	000	OP0	OP1			OP0			
	001	—	OP1			OP1			
	010	OP2	OP3			OP2			
	100	OP4	OP5			OP4			
	101	—	OP5			OP5			
	110	OP6	OP7			OP6			
Word	000	OP0	OP1			OP0			
	100	OP4	OP5			OP4			

¹ Address state is the calculated address for port size.

11.5.4 Command Sequence Examples for NAND Flash EEPROM

In order to program the eLBC and FCM for executing NAND Flash command sequences, command codes and pause states should be obtained from the relevant NAND Flash device data sheet and programmed into FCM configuration registers. This section illustrates some common sequences for large-page, multi-gigabit NAND Flash EEPROMs; however, details should be verified against manufacturers' specific programming data.

Throughout these examples it is assumed that one or more banks of eLBC has been configured under FCM control ($BR_n[\text{MSEL}] = 001$), with base address, port size, ECC mode, and timing parameters configured in accordance with the *MPC8309 PowerQUICC II Pro Integrated Communications Processor Hardware Specification*.

11.5.4.1 NAND Flash Soft Reset Command Sequence Example

An example of configuring FCM to execute a soft reset command to large-page NAND Flash is shown in [Table 11-42](#). This sequence does not require use of the shared FCM buffer RAM. The sequence is initiated by writing FMR[OP] = 10, and issuing a special operation to the bank. At the conclusion of the sequence, eLBC will issue a command complete interrupt (LTESR[CC]) if interrupts are enabled.

Table 11-42. FCM Register Settings for Soft Reset (OR_n[PGS] = 1)

Register	Initial Contents	Description
FCR	0xFF00_0000	CMD0 = 0xFF = reset command; other commands unused
FBAR	—	Unused
FPAR	—	Unused
FBCR	—	Unused
MDR	—	Unused
FIR	0x4000_0000	OP0 = CM0 = command 0; OP1–OP7 = NOP

11.5.4.2 NAND Flash Read Status Command Sequence Example

An example of configuring FCM to execute a status read command to large-page NAND Flash is shown in [Table 11-43](#). This sequence does not require use of the shared FCM buffer RAM, but reads the NAND Flash status into register MDR[AS0]. The sequence is initiated by writing FMR[OP] = 10 and issuing a special operation to the bank. At the conclusion of the sequence, eLBC will issue a command complete interrupt (LTESR[CC]) if interrupts are enabled.

Table 11-43. FCM Register Settings for Status Read (OR_n[PGS] = 1)

Register	Initial Contents	Description
FCR	0x7000_0000	CMD0 = 0x70 = read status command; other commands unused
FBAR	—	Unused
FPAR	—	Unused
FBCR	—	Unused
MDR	—	Status returned in AS0
FIR	0x4B00_0000	OP0 = CM0 = command 0; OP1 = RS = read status to MDR; OP2–OP7 = NOP

11.5.4.3 NAND Flash Read Identification Command Sequence Example

An example of configuring FCM to execute a status ID command to large-page NAND Flash is shown in [Table 11-44](#). This sequence does not require use of the shared FCM buffer RAM, but uses MDR to set up a dummy address prior to the sequence, and then to receive the first 4 bytes of ID during the sequence. The sequence is initiated by writing FMR[OP] = 10, and issuing a special operation to the bank. At the

conclusion of the sequence, eLBC will issue a command complete interrupt (LTESR[CC]) if interrupts are enabled. MDR[AS3–AS0] then can be read to obtain the first 4 bytes of NAND Flash ID.

Table 11-44. FCM Register Settings for ID Read (ORn[PGS] = 1)

Register	Initial Contents	Description
FCR	0x9000_0000	CMD0 = 0x90 = read ID command; other commands unused
FBAR	—	Unused
FPAR	—	Unused
FBCR	—	Unused
MDR	0x0000_0000	AS0 = 0x00 = dummy address for read ID command; AS0–AS3 return with first 4 bytes of ID code
FIR	0x43BB_BBB0	OP0 = CM0 = command 0; OP1 = UA = user address from MDR; OP2–OP6 = RS = read 4 bytes ID into MDR[AS3–AS0]; OP7 = NOP

11.5.4.4 NAND Flash Page Read Command Sequence Example

An example of configuring FCM to execute a random page read command to large-page NAND Flash is shown in [Table 11-45](#). This sequence reads an entire page (main and spare region) into the shared FCM buffer RAM, checking ECC as it proceeds. The sequence is initiated by writing FMR[OP] = 11, and issuing a special operation to the bank. At the conclusion of the sequence, eLBC will issue a command complete interrupt (LTESR[CC]) if interrupts are enabled.

Table 11-45. FCM Register Settings for Page Read (ORn[PGS] = 1)

Register	Initial Contents	Description
FCR	0x0030_0000	CMD0 = 0x00 = random read address entry; CMD1 = 0x30 = read page
FBAR	block index (for example block 0x0001_0AB4)	BLK locates index of 128-Kbyte block
FPAR	page offset (for example 0x0000_5000 locates page 5, buffer 1)	PI locates page index in BLK; PI mod 2 indexes FCM buffer RAM; MS = 0 and CI = 0
FBCR	0x0000_0000	BC = 0 to read entire 2112-byte page with ECC check
MDR	—	Unused
FIR	0x4125_E000	OP0 = CM0 = command 0; OP1 = CA = column address; OP2 = PA = page address; OP3 = CM1 = command 1; OP4 = RBW = wait on Flash ready and read data into FCM buffer; OP5–OP7 = NOP

11.5.4.5 NAND Flash Block Erase Command Sequence Example

An example of configuring FCM to execute a block erase command to large-page NAND Flash is shown in [Table 11-46](#). This sequence does not require use of the shared FCM buffer RAM, but returns with the erase status in MDR[AS0]. The sequence is initiated by writing FMR[OP] = 11, and issuing a special operation to the bank. At the conclusion of the sequence, eLBC will issue a command complete interrupt (LTER[CC]) if interrupts are enabled.

Note that operations specified by OP3 and OP4 (status read) should never be skipped while erasing a NAND Flash device, because, in case that happens, contention may arise on LGPL4. A possible case is that the next transaction from eLBC may try to use that pin as an output and since the NAND Flash device might already be driving it, contention will occur. In case OP3 and OP4 operations are skipped, it may also happen that a new command is issued to the NAND Flash device even when the device has not yet finished processing the previous request. This may also result in unpredictable behavior.

Table 11-46. FCM Register Settings for Block Erase (ORn[PGS] = 1)

Register	Initial Contents	Description
FCR	0x6070_D000	CMD0 = 0x60 = block address entry; CMD1 = 0x70 = read status CMD2 = 0xD0 = erase block;
FBAR	Block Index (for example block 0x0001_0AB4)	BLK locates index of 128-Kbyte block
FPAR	0x0000_0000	PI = 0 to locate block boundary
FBCR	—	Unused
MDR	—	Returns with AS0 holding erase status
FIR	0x426D_B000	OP0 = CM0 = command 0; OP1 = PA = page address; OP2 = CM2 = command 2; OP3 = CW1 = wait on Flash ready and issue command 1; OP4 = RS = read erase status into MDR[AS0]; OP5–OP7 = NOP

11.5.4.6 NAND Flash Program Command Sequence Example

An example of configuring FCM to execute a program command to large-page NAND Flash is shown in [Table 11-47](#). This sequence writes an entire page (main and spare region) from the shared FCM buffer RAM, generating ECC as it proceeds. The shared buffer (buffer 1 for page index 5) must be initialized by software prior to starting the sequence. The sequence is initiated by writing FMR[OP] = 11, and issuing a special operation to the bank. At the conclusion of the sequence, eLBC will issue a command complete interrupt (LTER[CC]) if interrupts are enabled. The status of the programming operation is returned in MDR[AS0].

Note that operations specified by OP5 and OP6 (status read) should never be skipped while programming a NAND Flash device, because, in case that happens, contention may arise on LGPL4. A possible case is that the next transaction from eLBC may try to use that pin as an output and since the NAND Flash device might already be driving it, contention will occur. In case OP5 and OP6 operations are skipped, it may also

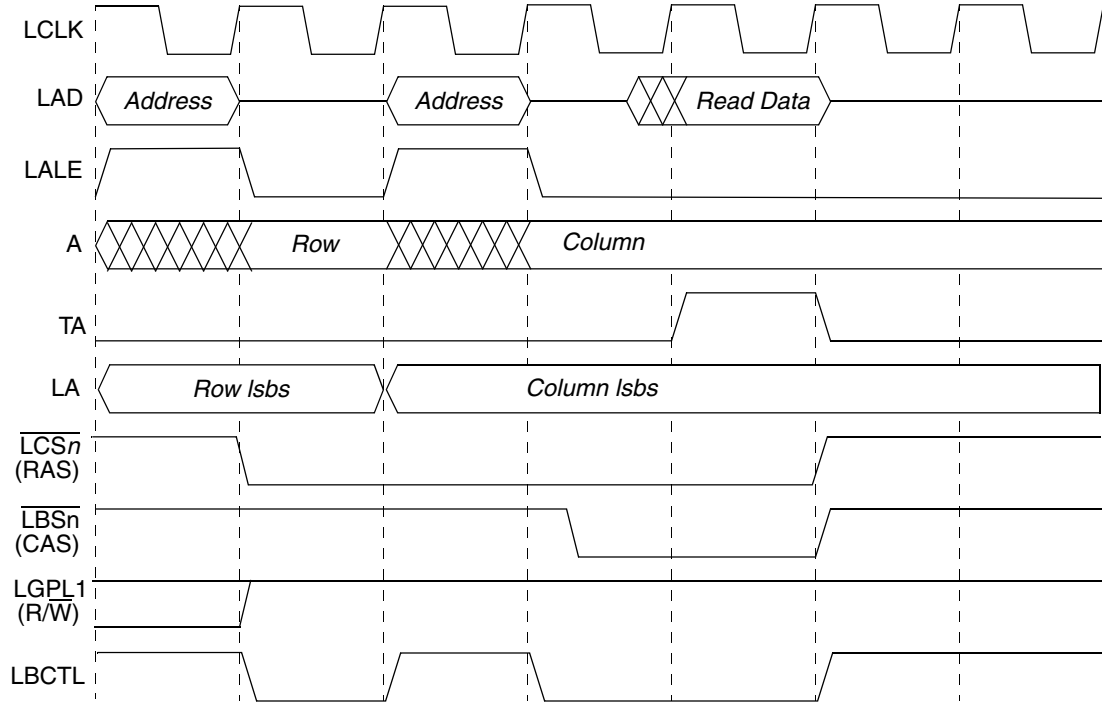
happen that a new command is issued to the NAND Flash device even when the device has not yet finished processing the previous request. This may also result in unpredictable behavior.

Table 11-47. FCM Register Settings for Page Program (OR n [PGS] = 1)

Register	Initial Contents	Description
FCR	0x8070_1000	CMD0 = 0x80 = page address and data entry; CMD1 = 0x70 = read status CMD2 = 0x10 = program page;
FBAR	block index (for example block 0x0001_0AB4)	BLK locates index of 128-Kbyte block
FPAR	page offset (for example 0x0000_5000 locates page 5, buffer 1)	PI locates page index in BLK; PI mod 2 indexes FCM buffer RAM; MS = 0 and CI = 0
FBCR	0x0000_0000	BC = 0 to write entire 2112-Byte page with ECC generation
MDR	—	Returns with AS0 holding program status
FIR	0x4128_6DB0	OP0 = CM0 = command 0; OP1 = CA = column address; OP2 = PA = page address; OP3 = WB = write data from buffer; OP4 = CM2 = command 2; OP5 = CW1 = wait on Flash ready and issue command 1; OP6 = RS = read erase status into MDR[AS0]; OP7 = NOP

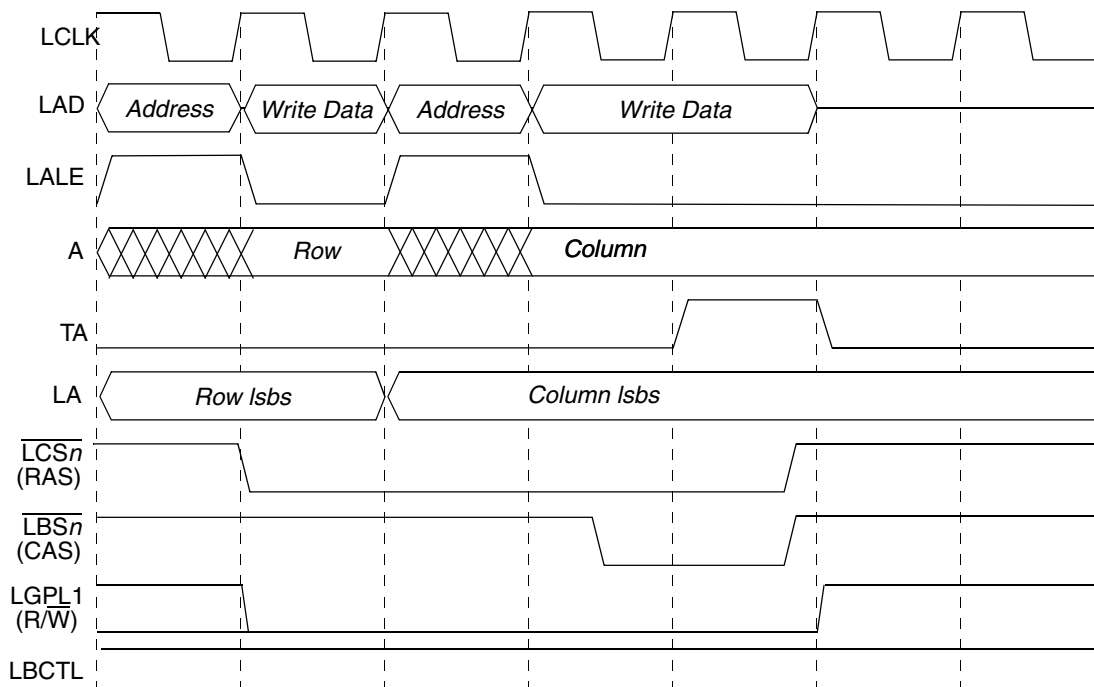
11.5.5 Interfacing to Fast-Page Mode DRAM Using UPM

Connecting the local bus UPM controller to a DRAM device requires a detailed examination of the timing diagrams representing the possible memory cycles that must be performed when accessing this device. This section describes timing diagrams for various UPM configurations for fast-page mode DRAM, with LCRR[CLKDIV] = 4 or 8. The illustrative examples shown in [Figure 11-72](#)–[Figure 11-77](#) may not represent the timing necessary for any specific device used with the eLBC. Here, LGPL1 is programmed to drive R/W of the DRAM, although any LGPL n signal may be used for this purpose.



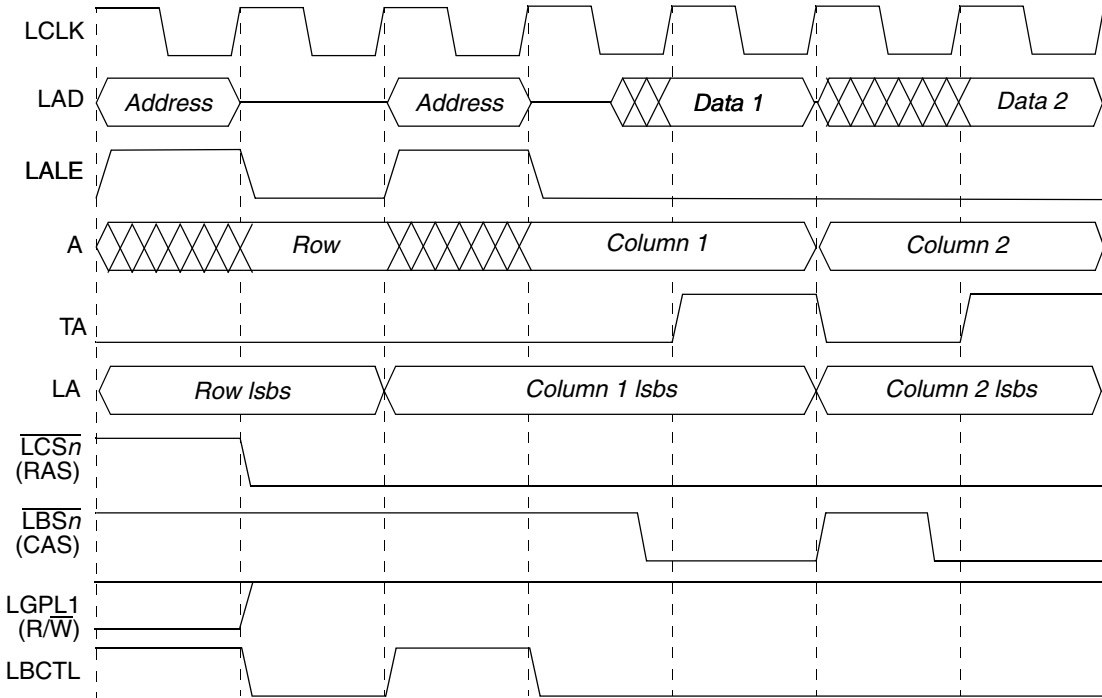
cst1	0	LALE pause (due to change in AMX)	0	0	Bit 0
cst2	0		0	0	Bit 1
cst3	0		0	0	Bit 2
cst4	0		0	0	Bit 3
bst1	1		1	0	Bit 4
bst2	1		0	0	Bit 5
bst3	1		0	0	Bit 6
bst4	1		0	0	Bit 7
g0i0					Bit 8
g0i1					Bit 9
g0h0					Bit 10
g0h1					Bit 11
g1t1	1		1	1	Bit 12
g1t3	1		1	1	Bit 13
g2t1					Bit 14
g2t3					Bit 15
g3t1					Bit 16
g3t3					Bit 17
g4t1					Bit 18
g4t3					Bit 19
g5t1					Bit 20
g5t3					Bit 21
redo[0]					Bit 22
redo[1]					Bit 23
loop	0		0	0	Bit 24
exen	0		0	0	Bit 25
amx0	1		0	0	Bit 26
amx1	0		0	0	Bit 27
na	0		0	0	Bit 28
uta	0		0	1	Bit 29
todt	0		0	1	Bit 30
last	0		0	1	Bit 31
	RSS	RSS + 1	RSS + 1	RSS + 2	

Figure 11-72. Single-Beat Read Access to FPM DRAM



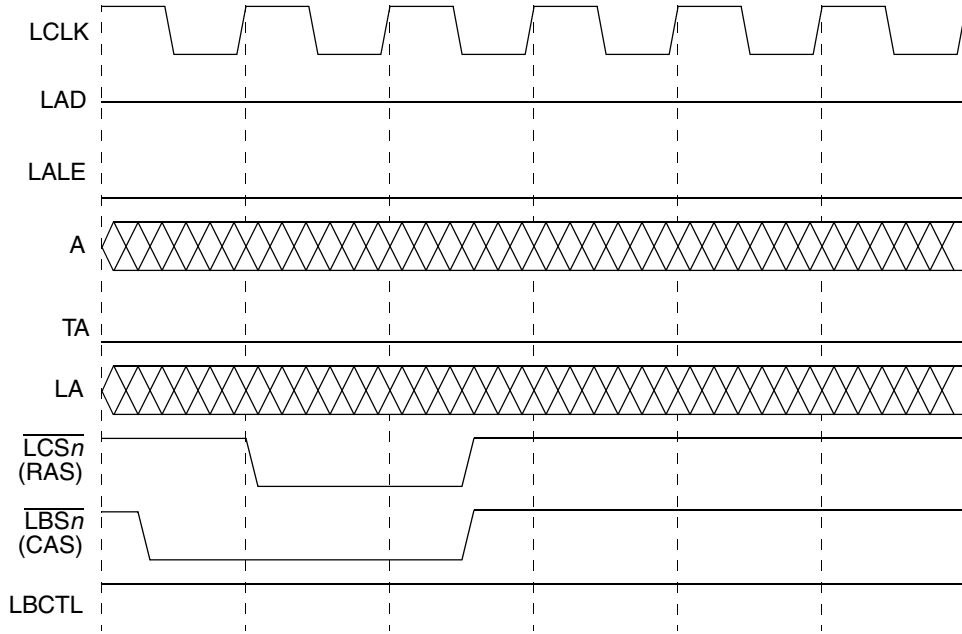
cst1	0	LALE pause (due to change in AMX)	0	0	Bit 0
cst2	0		0	0	Bit 1
cst3	0		0	0	Bit 2
cst4	0		0	1	Bit 3
bst1	1		1	0	Bit 4
bst2	1		1	0	Bit 5
bst3	1		0	0	Bit 6
bst4	1		0	1	Bit 7
g0i0					Bit 8
g0i1					Bit 9
g0h0					Bit 10
g0h1					Bit 11
g1t1	0		0	0	Bit 12
g1t3	0		0	0	Bit 13
g2t1					Bit 14
g2t3					Bit 15
g3t1					Bit 16
g3t3					Bit 17
g4t1					Bit 18
g4t3					Bit 19
g5t1					Bit 20
g5t3					Bit 21
redo[0]					Bit 22
redo[1]					Bit 23
loop	0		0	0	Bit 24
exen	0		0	0	Bit 25
amx0	1		0	0	Bit 26
amx1	0		0	0	Bit 27
na	0		0	0	Bit 28
uta	0		0	1	Bit 29
todt	0		0	1	Bit 30
last	0		0	1	Bit 31
		WSS	WSS + 1	WSS + 1	WSS + 2

Figure 11-73. Single-Beat Write Access to FPM DRAM



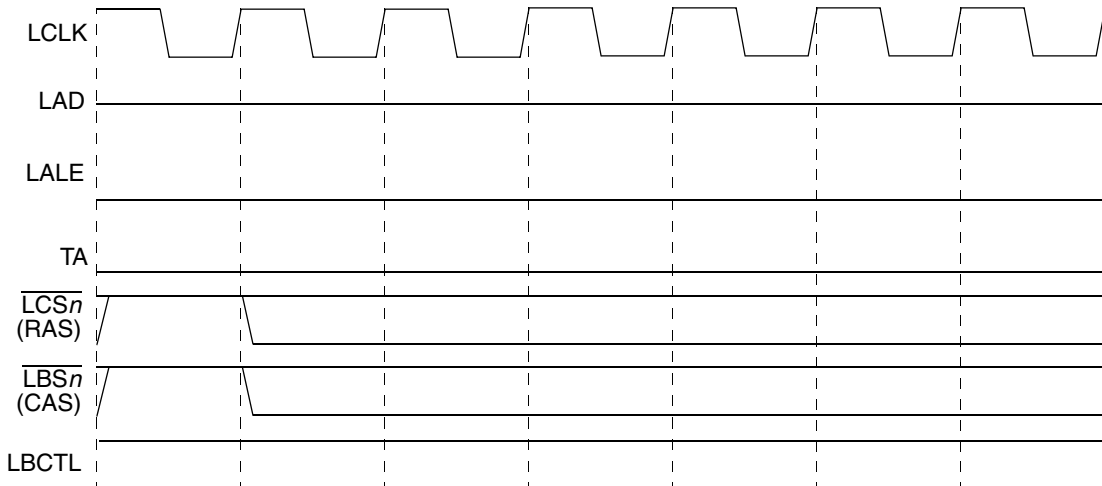
cst1	0	LALE pause (due to change in AMX)	0	0	1	Bit 0
cst2	0		0	0	1	Bit 1
cst3	0		0	0	1	Bit 2
cst4	0		0	0	1	Bit 3
bst1	1		1	0	1	Bit 4
bst2	1		1	0	1	Bit 5
bst3	1		1	0	1	Bit 6
bst4	1		0	0	1	Bit 7
g0l0						Bit 8
g0l1						Bit 9
g0h0						Bit 10
g0h1						Bit 11
g1t1	1		1	1	1	Bit 12
g1t3	1		1	1	1	Bit 13
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 11-74. 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 11-75. Refresh Cycle (CBR) to FPM DRAM



cst1	1	Bit 0
cst2	1	Bit 1
cst3	1	Bit 2
cst4	1	Bit 3
bst1	1	Bit 4
bst2	1	Bit 5
bst3	1	Bit 6
bst4	1	Bit 7
g0l0		Bit 8
g0l1		Bit 9
g0h0		Bit 10
g0h1		Bit 11
g1t1		Bit 12
g1t3		Bit 13
g2t1		Bit 14
g2t3		Bit 15
g3t1		Bit 16
g3t3		Bit 17
g4t1		Bit 18
g4t3		Bit 19
g5t1		Bit 20
g5t3		Bit 21
redo[0]		Bit 22
redo[1]		Bit 23
loop	0	Bit 24
exen	0	Bit 25
amx0	0	Bit 26
amx1	0	Bit 27
na	0	Bit 28
uta	0	Bit 29
todt	1	Bit 30
last	1	Bit 31
EXS		

Figure 11-76. Exception Cycle

11.5.6 Interfacing to ZBT SRAM Using UPM

ZBT SRAMs have been designed to optimize the performance of table access in networking applications. This section describes how to interface to ZBT SRAMs. [Figure 11-77](#) shows the connections. The UPM is used to generate control signals. The same interfacing is used for pipelined and flow-through versions of ZBT SRAMs. However different UPM patterns must be generated for those cases. As ZBT SRAMs are

mostly used by performance-critical applications, it is assumed here that, typically, the maximum width of the local bus of 16 bits will be used.

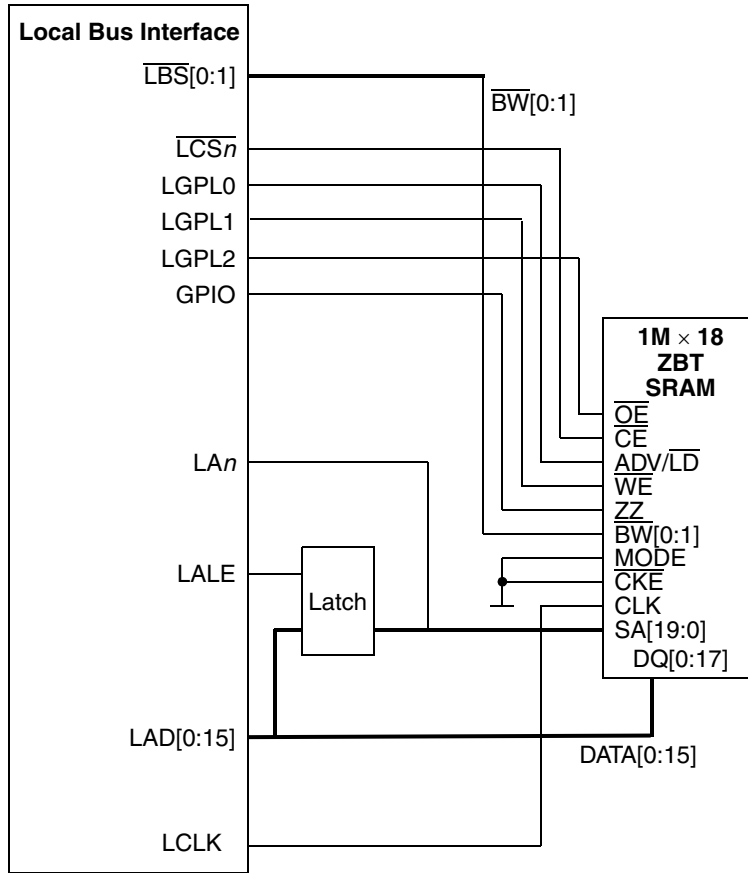


Figure 11-77. Interface to ZBT SRAM

ZBT SRAMs allow different configurations. For the local bus, the burst order should be set to linear burst order by tying the mode pin to GND. \overline{CKE} should also be tied to ground.

ZBT SRAMs perform four-beat bursts. Because the eLBC generates sixteen-beat transactions (for 16-bit ports) the UPM breaks down each burst into four consecutive four-beat bursts. The internal address generator of the eLBC generates the new $\{A21, A22\}$ for the second, third, and fourth burst. In other words, because linear burst is used on the SRAM, the device itself bursts with the burst addresses of $[0:1:2:3]$. The local bus always generates linear bursts and expects $[0:1:2:3:4:5:6:7:\dots:15]$. Therefore, four consecutive linear bursts of the ZBT SRAM with $\{A21, A22\} = \{0,0\}$ for the first burst, $\{A21, A22\} = \{0,1\}$ for the second burst, $\{A21, A22\} = \{1,0\}$ for the third burst, and $\{A21, A22\} = \{1,1\}$ for the fourth burst give the desired burst pattern.

The UPM also supports single beat accesses. Because the ZBT SRAM does not support this and always responds with a burst, the UPM pattern has to take care that data for the critical beat is provided (for write) or sampled (for read), and that the rest of the burst is ignored (by negating \overline{WE}). The UPM controller basically has to wait for the end of the SRAM burst to avoid bus contention with further bus activities.



Chapter 12

Enhanced Secure Digital Host Controller

12.1 Overview

The enhanced secure digital host controller (eSDHC) provides an interface between the host system and these types of memory cards:

- Multi Media card (MMC)

MMC is a universal low-cost data storage and communication medium designed to cover a wide area of applications including mobile video and gaming, which are available from either pre-loaded MMCs or downloadable from cellular phones, WLAN, or other wireless networks.

- Secure digital (SD) card

The secure digital (SD) card is an evolution of old MMC technology. It is specifically designed to meet the security, capacity, performance, and environment requirements inherent in the emerging audio and video consumer electronic devices. The physical form factor, pin assignments, and data transfer protocol are forward-compatible with the old MMC.

- SDIO

Under the SD protocol, the SD cards can be categorized as a memory card, I/O card, or combo card. The memory card invokes a copyright protection mechanism that complies with the security of the SDMI standard. The I/O card provides high-speed data I/O with low power consumption for mobile electronic devices. The combo card has both memory and I/O functions. To keep [Figure 12-1](#) simple, the diagram does not show cards with reduced sizes.

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

transmission level. [Figure 12-1](#) shows connection of the eSDHC.

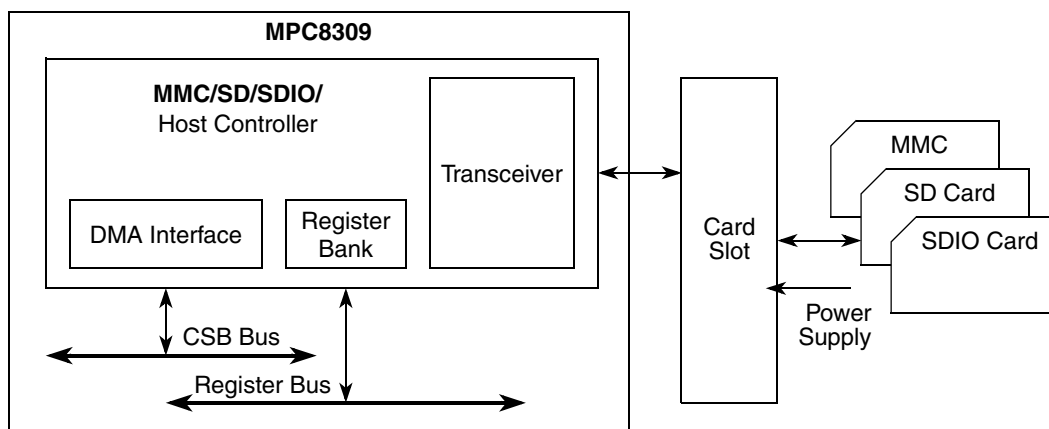


Figure 12-1. System Connection of the eSDHC

Figure 12-2 is a block diagram of the eSDHC.

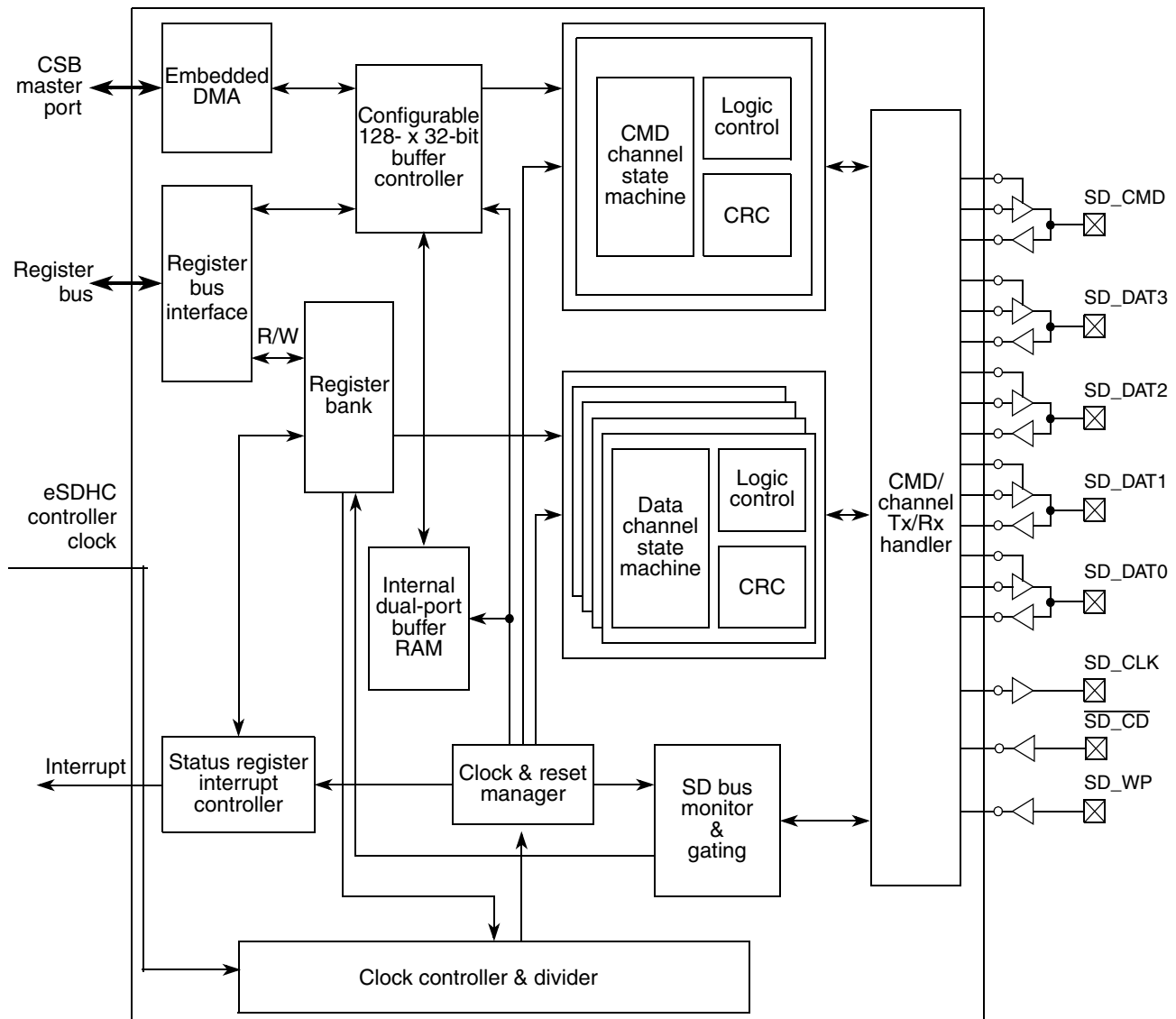


Figure 12-2. eSDHC Block Diagram

12.2 Features

The eSDHC includes the following features:

- Compatible with the following specifications:
 - *SD Host Controller Standard Specification, Version 2.0* (<http://www.sdcard.org>) with test event register support
 - *MultiMedia Card System Specification, Version 4.0* (<http://www.mmca.org>)
 - *SD Memory Card Specification, Version 2.0* (<http://www.sdcard.org>)
 - *SD Specifications, Part 1, Physical Layer Specification, Version 2.0*

- *SD Card Specification, Part E1, SD Input/Output (SDIO) Card Specification, Version 2.0*
- Designed to work with SD Memory, miniSD Memory, SDIO, miniSDIO, SD Combo, SDIO, MMC, MMC*plus*, and RS-MMCs
- Card bus clock frequency up to 33.25 MHz
- Supports 1-/4-bit SD and SDIO modes, 1-/4-bit MMC modes
 - Up to 133 Mbps data transfer for SD cards/SDIO/MMCs using four parallel data lines
- Single- and multi-block read and write
- Supports block sizes of 1 ~ 4096 bytes
- Write-protection switch for write operations
- Synchronous abort
- Pause during the data transfer at a block gap
- SDIO read wait and suspend/resume operations
- Auto CMD12 for multi-block transfer
- Host can initiate non-data transfer commands while the data transfer is in progress
- Allows cards to interrupt the host in 1- and 4-bit SDIO modes
- Supports interrupt period, defined in the SDIO standard
- Fully configurable 128 × 32-bit FIFO for read/write data
- DMA capabilities

12.2.1 Data Transfer Modes

The eSDHC can select the following modes for data transfer:

- SD 1-bit
- SD 4-bit
- MMC 1-bit
- MMC 4-bit
- Identification mode (upto 400 KHz)
- Full-speed mode (up to 25 MHz) or high-speed mode (up to 33.25 MHz)

12.3 External Signal Description

The eSDHC has eight chip I/O signals.

- SD_CLK is the internally generated clock signal that drives the MMC, SDIO, or SD card.
- SD_CMD I/O sends commands and receives responses from the card.
- SD_DAT3–SD_DAT0 performs data transfers between the eSDHC and the card.
- SD_ $\overline{\text{CD}}$ and SD_WP are card detection and write protection signals from the socket.
 - Signals SD_ $\overline{\text{CD}}$ and SD_WP are optional for system implementation.

Table 12-1 shows the properties of the eSDHC I/O signals.

Table 12-1. Signal Properties

Name	Port	Function	Reset State	Pull up/Pull down Required
SD_CLK	O	Clock for MMC/SD/SDIO card	0	N/A
SD_CMD	I/O	Command line to card	High impedance	Pull up
SD_DAT3	I/O	4-bit mode: DAT3 line or configured as card detection pin 1-bit mode: May be configured as card detection pin	High impedance	Board should have 100-K pull down. The card drives 50-K pull up as required by the <i>SD Card Specification</i> .
SD_DAT2	I/O	4-bit mode: DAT2 line or read wait 1-bit mode: Read wait	High impedance	Pull up
SD_DAT1	I/O	4-bit mode: DAT1 line or interrupt detect 1-bit mode: Interrupt detect	High impedance	Pull up
SD_DAT0	I/O	DAT0 line or busy-state detect	High impedance	Pull up
$\overline{\text{SD_CD}}$	I	Card detection pin <ul style="list-style-type: none"> $\overline{\text{SD_CD}}$ = HIGH implies card is not present $\overline{\text{SD_CD}}$ = LOW implies card is present. 	N/A	Board should have 100-K pull up on the $\overline{\text{SD_CD}}$ signal and a 330 Ω pull down on the common pin of the Write Protect/Card Detect switch on the SD card connector.
SD_WP	I	Card write protect detect; if not used, tied low <ul style="list-style-type: none"> SD_WP = HIGH implies the write protect switch is enabled on the card (Writes disabled) SD_WP = LOW implies write protect switch is disabled on the card (Writes enabled) 	N/A	Board should have 100-K pull up on the SD_WP signal and a 330 Ω pull down on the common pin of the Write Protect/Card Detect switch on the SD card connector.

12.4 Memory Map/Register Definition

Table 12-2 shows the memory mapped registers of the eSDHC module and their offsets. It lists the offset, name, and a cross-reference to the complete description of each register. Note that the full register address is comprised of IMMRBAR together with the eSDHC block base address and offset listed in Table 12-2. Undefined 4-byte address spaces within offset 0x000–0xFFF are reserved.

NOTE

All eSDHC registers must be accessed as aligned 4-byte quantities.
Accesses to the eSDHC registers that are less than 4-bytes are not supported.

Table 12-2. eSDHC Memory Map

eSDHC Registers—Block Base Address 0x2_E000				
Offset	Register	Access	Reset	Section/Page
0x000	DMA system address (DSADDR)	R/W	0x0000_0000	12.4.1/12-7
0x004	Block attributes (BLKATTR)	R/W	0x0001_0000	12.4.2/12-7
0x008	Command argument (CMDARG)	R/W	0x0000_0000	12.4.3/12-8
0x00C	Command transfer type (XFERTYP)	R/W	0x0000_0000	12.4.4/12-9
0x010	Command response0 (CMDRSP0)	R	0x0000_0000	12.4.5/12-12
0x014	Command response1 (CMDRSP1)	R	0x0000_0000	12.4.5/12-12
0x018	Command response2 (CMDRSP2)	R	0x0000_0000	12.4.5/12-12
0x01C	Command response3 (CMDRSP3)	R	0x0000_0000	12.4.5/12-12
0x020	Data buffer access port (DATPORT)	R/W	0x0000_0000	12.4.6/12-14
0x024	Present state (PRSSTAT)	R	0x0F80_00F8	12.4.7/12-15
0x028	Protocol control (PROCTL)	R/W	0x0000_0020	12.4.8/12-19
0x02C	System control (SYSCTL)	Mixed	0x0000_8008	12.4.9/12-22
0x030	Interrupt status (IRQSTAT)	w1c	0x0000_0000	12.4.10/12-24
0x034	Interrupt status enable (IRQSTATEN)	R/W	0x117F_013F	12.4.11/12-29
0x038	Interrupt signal enable (IRQSIGEN)	R/W	0x0000_0000	12.4.12/12-31
0x03C	Auto CMD12 status (AUTO12ERR)	R	0x0000_0000	12.4.13/12-33
0x040	Host controller capabilities (HOSTCAPBLT)	R	0x01F3_0000	12.4.14/12-35
0x044 ¹	Watermark level (WML)	R/W	0x1010_1010	12.4.15/12-36
0x050	Force event (FEVT)	W	0x0000_0000	12.4.16/12-37
0x0FC	Host controller version (HOSTVER)	R	0x0000_1201	12.4.17/12-39
0x40C	DMA control register (DCR)	R/W	0x0000_0000	12.4.18/12-39

¹ The addresses following 0x044, except 0x050, 0x0FC and 0x40C, are reserved and read as all 0s. Writes to these registers are ignored.

NOTE

For details on programming the CSB/eSDHC interface, see [Section 6.3.2.11, “eSDHC Control Register \(SDHCCR\).”](#)

12.4.1 DMA System Address Register (DSADDR)

The DMA system address register contains the system memory address used for DMA transfers. Only access this register when no transactions are executing (after transactions have stopped). The host driver should wait until PRSSTAT[DLA] is cleared.

Figure 12-3 shows the DMA system address register.

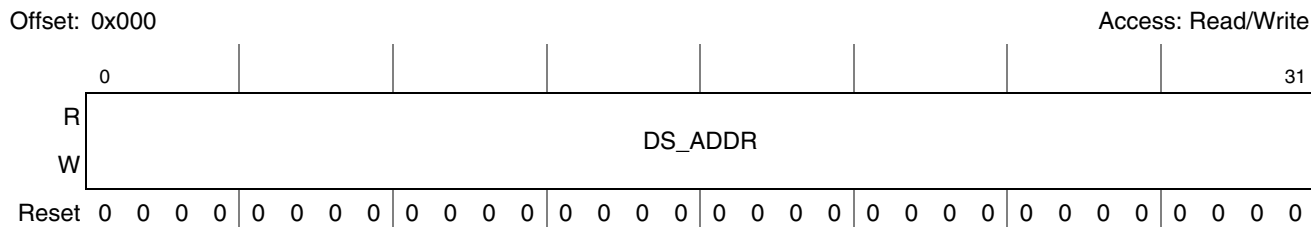


Figure 12-3. DMA System Address Register (DSADDR)

Table 12-3 describes the DSADDR fields.

Table 12-3. DSADDR Field Descriptions

Bit	Name	Description
0–31	DS_ADDR	DMA system address. When the eSDHC stops a DMA transfer, this register points to the system address of the next contiguous data position. Note: The DS_ADDR must be aligned to a four-byte boundary; the two least-significant bits must be cleared.

12.4.2 Block Attributes Register (BLKATTR)

The block attributes register configures the number of data blocks and the number of bytes in each block. Only access this register when no transactions are executing (after transactions have stopped). The host driver should wait until PRSSTAT[DLA] is cleared. During a data transfer, the following may occur:

- Reading this register may return an invalid value.
- Writing this register is ignored.

Figure 12-4 shows the DMA system address register.

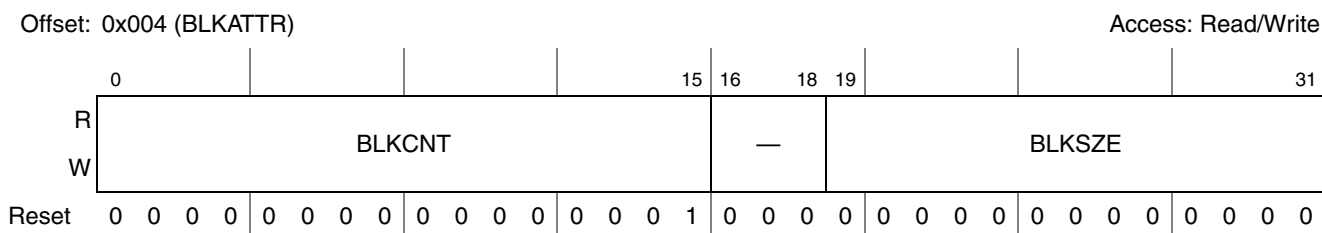


Figure 12-4. Block Attributes Register (BLKATTR)

Table 12-4 describes the BLKATTR fields.

Table 12-4. BLKATTR Field Descriptions

Bit	Name	Description
0–15	BLKCNT	Block count for current transfer. This field is enabled when XFERTYP[BCEN] is set and is valid only for multiple block transfers. The host driver should set this field to a value between 1 and the maximum block count. The eSDHC decrements the block count after each block transfer and stops when the count reaches zero. Clearing this field results in no data blocks being transferred. When saving transfer context as a result of a suspend command, this field indicates the number of blocks yet to be transferred. When restoring transfer context prior to issuing a resume command, the host driver should write the previously saved block count. 0000 Stop count 0001 1 block 0002 2 blocks ... FFFF 65,535 blocks
16–18	—	Reserved
19–31	BLKSIZE	Transfer block size. Specifies the block size for block data transfers. Values can range from one byte up to the maximum buffer size. The DMA always writes at least four bytes to memory. Thus, software should allocate a buffer space rounded up to a 4-byte aligned size in order to avoid data corruption. 0000 No data transfer 0001 1 byte 0002 2 bytes 0003 3 bytes 0004 4 bytes ... 01FF 511 bytes 0200 512 bytes ... 0800 2048 bytes 1000 4096 bytes

12.4.3 Command Argument Register (CMDARG)

The command argument register, shown in Figure 12-5, contains the SD/MMC command argument.

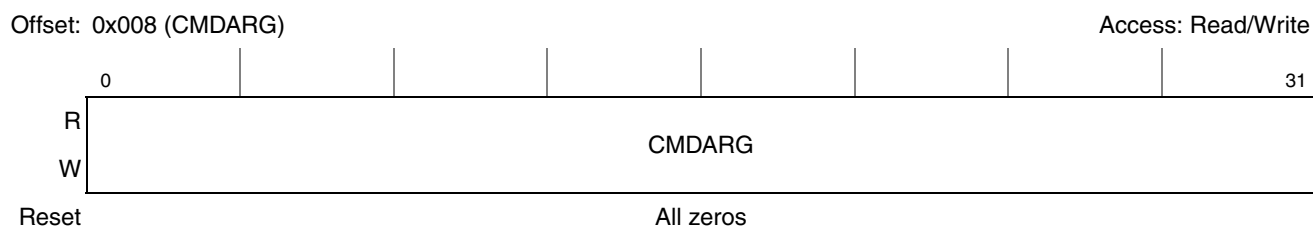


Figure 12-5. Command Argument Register (CMDARG)

Table 12-5 describes the CMDARG fields.

Table 12-5. CMDARG Field Descriptions

Bit	Name	Description
0–31	CMDARG	Command argument. The SD/MMC command argument is specified as bits 39–8 of the command format in the <i>SD or MMC Specification</i> . If PRSSTAT[CIHB] is set, this register is write-protected.

12.4.4 Transfer Type Register (XFERTYP)

The transfer type register, shown in Figure 12-6, controls the operation of data transfers. The host driver should set this register before issuing a command followed by a data transfer, or before issuing a resume command. To prevent data loss, the eSDHC prevents a write to the bits that are involved in the data transfer of this register while the data transfer is active.

The host driver should check PRSSTAT[CDIHB] and PRSSTAT[CIHB] before writing to this register.

- If PRSSTAT[CDIHB] is set, any attempt to send a command with data by writing to this register is ignored.
- If PRSSTAT[CIHB] is set, any write to this register is ignored.

Offset: 0x00C (XFERTYP)

Access: Read/Write

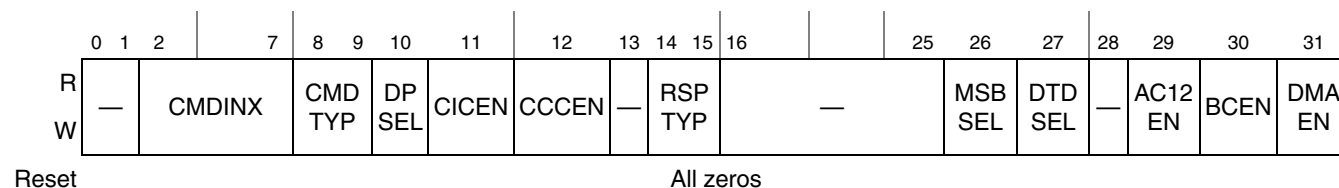


Figure 12-6. Transfer Type Register (XFERTYP)

Table 12-6 describes the XFERTYP fields.

Table 12-6. XFERTYP Field Descriptions

Bit	Name	Description
0–1	—	Reserved
2–7	CMDINX	Command index. These bits should be set to the command number (CMD0–63, ACMD0–63) that is specified in bits 45–40 of the command format in the <i>SD Memory Card Physical Layer Specification</i> and <i>SDIO Card Specification</i> .

Table 12-6. XFERTYP Field Descriptions (continued)

Bit	Name	Description
8–9	CMDTYP	<p>Command type. There are three types of special commands: suspend, resume, and abort. Clear this bit field for all other commands.</p> <ul style="list-style-type: none"> • Suspend command. If the suspend command succeeds, the eSDHC assumes the SD bus has been released and it is possible to issue the next command which uses the SD_DAT line. The eSDHC de-asserts read wait for read transactions and stops checking busy for write transactions. In 4-bit mode, the interrupt cycle starts. If the suspend command fails, the eSDHC maintains its current state, and the host driver should restart the transfer by setting PROCTL[CREQ]. The eSDHC does not check if the suspend command succeeds or not. It is the host driver's responsibility to issue a normal CMD52 marked as suspend command when the suspend request is accepted by the card, so that eSDHC can be informed that the SD bus is released and de-assert read wait during read operation. • Resume command. The host driver restarts the data transfer by restoring the registers saved before sending the suspend command and sends the resume command. The eSDHC checks for pending busy state before starting write transfers. • Abort command. If this command is set when executing a read transfer, the eSDHC stops reads to the buffer. If this command is set when executing a write transfer, the eSDHC stops driving the SD_DAT line. After issuing the abort command, the host driver should issue a software reset. (Abort transaction) <p>00 Normal—other commands 01 Suspend—CMD52 for writing bus suspend in the common card control register (CCCR), defined in the <i>SDIO Specification</i> 10 Resume—CMD52 for writing function select in CCCR 11 Abort—CMD12, CMD52 for writing I/O abort in CCCR</p>
10	DPSEL	<p>Data present select. Set to indicate that data is present and should be transferred using the SD_DAT line. It is cleared for the following:</p> <ul style="list-style-type: none"> • Commands using only the SD_CMD line (e.g. CMD52) • Commands with no data transfer but using busy signal on the SD_DAT[0] line (R1b or R5b, e.g. CMD38) <p>Note: In resume command, this bit should be set while the other bits in this register should be set the same as when the transfer initially launched.</p> <p>0 No data present 1 Data present</p>
11	CICEN	<p>Command index check enable.</p> <p>0 Disable. The index field is not checked. 1 Enable. The eSDHC checks the index field in the response to see if it has the same value as the command index. If it is not, it is reported as a command index error.</p>
12	CCCEN	<p>Command CRC check enable. The number of bits checked by the CRC field value changes according to the length of the response. (Refer to RSPTYP[1:0] and Table 12-8.)</p> <p>0 Disable. The CRC field is not checked. 1 Enable. The eSDHC checks the CRC field in the response if it contains the CRC field. If an error is detected, it is reported as a command CRC error.</p>
13	—	Reserved

Table 12-6. XFERTYP Field Descriptions (continued)

Bit	Name	Description
14–15	RSPTYP	Response type select. 00 No response 01 Response length 136 10 Response length 48 11 Response length 48 check busy after response
16–25	—	Reserved
26	MSBSEL	Multi/single block select. Enables multiple block SD_DAT line data transfers. For any other commands, this bit should be cleared. If this bit is cleared, it is not necessary to set the block count register. (Refer to Table 12-7 .) 0 Single block 1 Multiple blocks
27	DTDSEL	Data transfer direction select. Defines the direction of SD_DAT line data transfers. The bit is set by the host driver to transfer data from the SD card to the eSDHC and it is cleared for all other commands. 0 Write (host to card) 1 Read (card to host)
28	—	Reserved
29	AC12EN	Auto CMD12 enable. Multiple block transfers for memory require CMD12 to stop the transaction. If this bit is set, the eSDHC issues CMD12 automatically when the last block transfer is completed. The host driver should not set this bit to issue commands that do not require CMD12 to stop a multiple block data transfer. In particular, secure commands defined in the Part 3 File Security specification do not require CMD12. In a single block transfer, the eSDHC ignores this bit. 0 Disable 1 Enable
30	BCEN	Block count enable. Enables the block attributes register, which is only relevant for multiple block transfers. When this bit is cleared, the block attributes register is disabled, which is useful in executing an infinite transfer. 0 Disable 1 Enable
31	DMAEN	DMA enable. Enables DMA functionality as described in Section 12.5.2, “DMA CSB Interface.” If this bit is set, a DMA operation should begin when the host driver writes to the CMDINX field of the transfer type register. 0 Disable 1 Enable

[Table 12-7](#) shows how register settings determine types of data transfers.

Table 12-7. Determination of Transfer Type

Multi/Single Block Select XFERTYP[MSBSEL]	Block Count Enable XFERTYP[BCEN]	Block Count BLKATTR[BLKCNT]	Function
0	Don't Care	Don't Care	Single Transfer
1	0	Don't Care	Infinite Transfer
1	1	Positive Number	Multiple Transfer
1	1	Zero	No Data Transfer

Table 12-8 shows how the response type can be determined by the command index check enable, command CRC check enable, and response type bits.

Table 12-8. Relation Between Parameters and Name of Response Type

Response Type XFERTYP[RSPTYP]	Index Check Enable XFERTYP[CICEN]	CRC Check Enable XFERTYP[CCEN]	Response Type
00	0	0	No Response
01	0	1	R2
10	0	0	R3, R4
10	1	1	R1, R5, R6, R7
11	1	1	R1b, R5b

NOTE

- In the *SDIO Specification*, response type notation of R5b is not defined. R5 includes R5b in the *SDIO Specification*. But R5b is defined to specify the eSDHC checks busy status after receiving a response. For example, usually CMD52 is used as R5 but the I/O abort command should be used as R5b.
- The CRC field for R3 and R4 is expected to be all 1s. The CRC check should be disabled for these response types.

12.4.5 Command Response 0–3 (CMDRSP0–3)

The command response registers, shown in Figure 12-7, store the four parts of the response bits from the card.

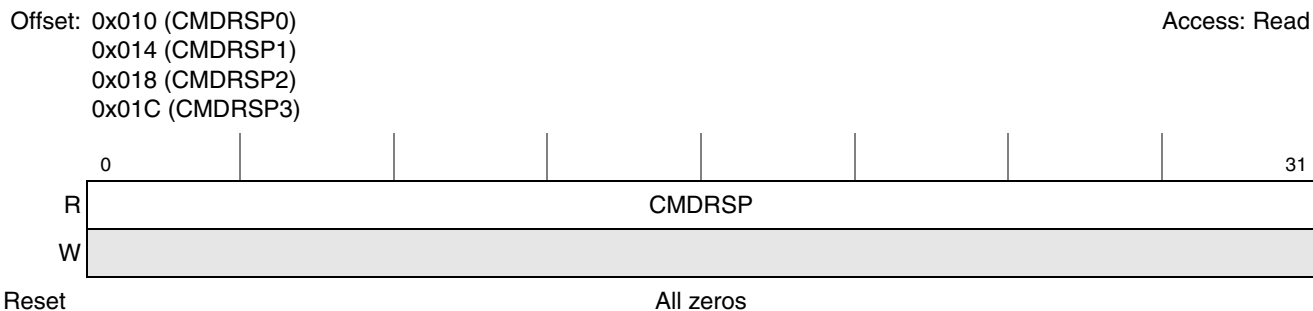


Figure 12-7. Command Response 0–3 Register (CMDRSP_n)

Table 12-9 describes the mapping of command responses from the SD bus to the command response registers for each response type. In the table, R[] refers to a bit range within the response data as transmitted on the SD bus.

Table 12-9. Response Bit Definition for Each Response Type

Response Type	Meaning of Response	Response Field	Response Register
R1, R1b (normal response)	Card status	R[39–8]	CMDRSP0
R1b (Auto CMD12 response)	Card status for Auto CMD12	R[39–8]	CMDRSP3
R2 (CID, CSD register)	CID/CSD register [127–8]	R[127–8]	{CMDRSP3[23:0], CMDRSP2, CMDRSP1, CMDRSP0}
R3 (OCR register)	OCR register for memory	R[39–8]	CMDRSP0
R4 (OCR register)	OCR register for I/O	R[39–8]	CMDRSP0
R5, R5b	SDIO response	R[39–8]	CMDRSP0
R6 (publish RCA)	New published RCA[31–16] and card status[15–0]	R[39–8]	CMDRSP0

This table shows that:

- Most responses with a length of 48 (R[47–0]) have 32 bits of the response data (R[39–8]) stored in the CMDRSP0 register.
- Responses of type R1b (Auto CMD12 responses) have response data bits R[39–8] stored in the CMDRSP3 register.
- Responses with length 136 (R[135–0]) have 120 bits of the response data (R[127–8]) stored in the CMDRSP0, 1, 2, and 3 registers.

To be able to read the response status efficiently, the eSDHC only stores part of the response data in the command response registers. This enables the host driver to efficiently read 32 bits of response data in one read cycle on a 32-bit bus system. Parts of the response, the index field, and the CRC are checked by the eSDHC (as specified by XFERTYP[CICEN, CCCEN]) and generate an error interrupt if any error is detected. The bit range for the CRC check depends on the response length. If the response length is 48, the eSDHC checks R[47–1], and if the response length is 136, the eSDHC checks R[119–1].

Since the eSDHC may have a multiple block data transfer executing concurrently with a CMD_wo_DAT command, the eSDHC stores the Auto CMD12 response in the CMDRSP3 register and the CMD_wo_DAT response is stored in CMDRSP0. This allows the eSDHC to avoid overwriting the Auto CMD12 response with the CMD_wo_DAT and vice versa. When the eSDHC modifies part of the command response registers it preserves the unmodified bits.

12.4.6 Buffer Data Port Register (DATPORT)

The buffer data port register, shown in [Figure 12-8](#), is a 32-bit data port register used to access the internal buffer.

NOTE

When the internal DMA is not enabled and a write transaction is in operation, DATPORT must not be read. DATPORT also must not be used to read (or write) data by the CPU if the data will be written (or read) by the eSDHC internal DMA.

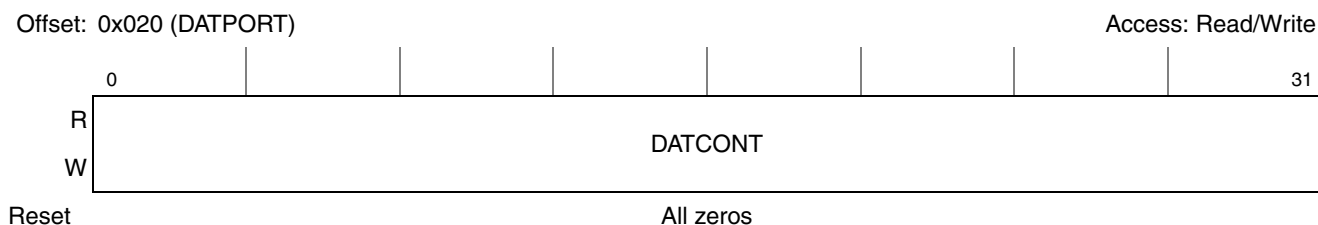


Figure 12-8. Buffer Data Port Register (DATPORT)

[Table 12-10](#) describes the DATPORT fields.

Table 12-10. DATPORT Field Descriptions

Bit	Name	Description
0–31	DATCONT	Data content. The buffer data port register is for 32-bit data access by the CPU. When the internal DMA is enabled, any write to this register is ignored, and a read from this register always yields 0.

12.4.7 Present State Register (PRSTAT)

The present state register (PRSTAT), shown in [Figure 12-9](#), indicates the status of the eSDHC to the host driver.

Offset: 0x024 (PRSTAT)

Access: Read

	0	3	4	7	8	9	11	12	13	14	15				
R	—			DLSL			CLSL	—			WPSPL	CDPL	—	CINS	
W															
Reset	0	0	0	0	1	1	1	1	0	0	0	0	0	0	
	16	19	20	21	22	23	24	25	26	27	28	29	30	31	
R	—			BREN	BWEN	RTA	WTA	SD OFF	PER OFF	HCK OFF	IPG OFF	SDSTB	DLA	CDIHB	CIHB
W															
Reset	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0

Figure 12-9. Present State Register (PRSTAT)

[Table 12-11](#) describes the PRSTAT fields.

Table 12-11. PRSTAT Field Descriptions

Bit	Name	Description										
0–3	—	Reserved										
4–7	DLSL	SD_DAT[3:0] line signal level. These bits are used to check the SD_DAT line level to recover from errors, and for debugging. This is especially useful in detecting the busy signal level from SD_DAT[0]. The reset value is affected by the external pull resistors. By default, read value of this bit field after reset is 0111, when SD_DAT[3] is pull-down and other lines are pull-up. <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>PRSTAT Bit</th> <th>SD_DATn</th> </tr> </thead> <tbody> <tr> <td>4</td> <td>3</td> </tr> <tr> <td>5</td> <td>2</td> </tr> <tr> <td>6</td> <td>1</td> </tr> <tr> <td>7</td> <td>0</td> </tr> </tbody> </table>	PRSTAT Bit	SD_DAT n	4	3	5	2	6	1	7	0
PRSTAT Bit	SD_DAT n											
4	3											
5	2											
6	1											
7	0											
8	CLSL	SD_CMD line signal level. This status is used to check the SD_CMD line level to recover from errors, and for debugging. The reset value is affected by the external pull resistor, by default, read value of this bit after reset is 1, when the command line is pull-up.										
9–11	—	Reserved										
12	WPSPL	Write protect switch pin level. The write protect switch is supported for memory and combo cards. This bit reflects the SD_WP pin of the card socket. A software reset does not affect this bit. The reset value is affected by the external write protect switch. If the SD_WP pin is not used, it should be tied to 0 so that the reset value of this bit is 0 and write is enabled. 0 Write protected (SD_WP = 1) 1 Write enabled (SD_WP = 0)										

Table 12-11. PRSSTAT Field Descriptions (continued)

Bit	Name	Description
13	CDPL	Card detect pin level. This bit reflects the inverse value of the SD $\overline{\text{CD}}$ pin for the card socket. Debouncing is not performed on this bit. This bit may be valid, but it is not guaranteed because of a propagation delay. Use of this bit is limited to testing since it must be debounced by software. A software reset does not affect this bit. Write to the force event register does not affect this bit. The reset value is affected by the external card detection pin. If this bit is not used, it should be tied to 0. 0 No card present (SD $\overline{\text{CD}}$ = 1) 1 Card present (SD $\overline{\text{CD}}$ = 0)
14	—	Reserved
15	CINS	Card inserted. Indicates if a card has been inserted. The eSDHC debounces this signal so that the host driver does not need to wait for it to stabilize. Changing from 0 to 1 generates a card-insertion interrupt in the interrupt status register and changing from 1 to 0 generates a card removal interrupt in the interrupt status register. A write to the force event register does not affect this bit. The software reset for all in the system control register does not affect this bit. A software reset does not affect this bit. 0 Power-on-reset or no card 1 Card inserted
16–19	—	Reserved
20	BREN	Buffer read enable. This status is used for non-DMA read transfers. The eSDHC allows for multiple data buffers in the internal memory. This read-only flag, when set, indicates that valid data greater than watermark level exists in the host-side buffer. When the buffer is read, this bit is cleared. When valid data greater than watermark level is ready in the buffer, this bit is set and a buffer read ready interrupt is generated (if the interrupt is enabled). 0 Buffer read disable 1 Buffer read enable
21	BWEN	Buffer write enable. This status is used for non-DMA write transfers. The eSDHC allows for multiple data buffers in the internal memory. This read-only flag, when set, indicates if space is available for greater than watermark level of write data. When the buffer is written, this bit is cleared. When the buffer can hold data greater than the watermark level, this bit is set and a buffer write ready interrupt is generated (if the interrupt is enabled). 0 Buffer write disable 1 Buffer write enable
22	RTA	Read transfer active. This status is used for detecting completion of a read transfer. This bit is set for either of the following conditions: <ul style="list-style-type: none"> • After the end bit of the read command • When writing a 1 to PROCTL[CREQ] to restart a read transfer This bit is cleared for either of the following conditions: <ul style="list-style-type: none"> • When the last data block as specified by block length is transferred to the system • When all valid data blocks have been transferred to the system and no current block transfers are being sent as a result of PROCTL[SABGREQ] being set. A transfer complete interrupt is generated when this bit changes to 0. 0 No valid data 1 Transferring data

Table 12-11. PRSSTAT Field Descriptions (continued)

Bit	Name	Description
23	WTA	<p>Write transfer active. This status indicates a write transfer is active. If this bit is 0, it means no valid write data exists in eSDHC.</p> <p>This bit is set in either of the following cases:</p> <ul style="list-style-type: none"> • After the end bit of the write command. • When writing a 1 to PROCTL[CREQ] to restart a write transfer. <p>This bit is cleared in either of the following cases:</p> <ul style="list-style-type: none"> • After getting the CRC status of the last data block, as specified by the transfer count (single and multiple) • After getting the CRC status of any block where data transmission is about to be stopped by a stop-at-block-gap request. <p>During a write transaction, a IRQSTAT[BGE] interrupt is generated when this bit is changed to 0, as result of PROCTL[SABGREQ] being set. This status is useful for the host driver in determining when to issue commands during write busy.</p> <p>0 No valid data 1 Transferring data</p>
24	SDOFF	<p>SD clock gated off internally. Indicates the SD clock is internally gated off because of a buffer overrun, buffer underrun, or a read pause without read-wait assertion. This bit is for the host driver to debug data transaction on SD bus.</p> <p>This status bit resets to 0, but reflects the value of the automatic clock gating and may transition to 1 if the eSDHC is idle.</p>
25	PEROFF	<p>The internal bus clock gated off internally. This status bit indicates the internal bus clock is internally gated off. This bit is for the host driver to debug a transaction on SD bus.</p> <p>This status bit resets to 0, but reflects the value of the automatic clock gating and may transition to 1 if the eSDHC is idle.</p>
26	HCKOFF	<p>Master clock gated off internally. This status bit indicates master clock is internally gated off. This bit is for the host driver to debug a data transfer.</p> <p>This status bit resets to 0, but reflects the value of the automatic clock gating and may transition to 1 if the eSDHC is idle.</p>
27	IPGOFF	<p>Controller clock gated off internally. Indicates that the controller clock is internally gated off. This bit is for the host driver to debug. The controller clock runs at $csb_clk / SCCR[ESDHCCM]$.</p> <p>This status bit resets to 0, but reflects the value of the automatic clock gating and may transition to 1 if the eSDHC is idle.</p>
28	SDSTB	<p>SD Clock Stable</p> <p>This status bit indicates that the internal card clock is stable. This bit is for the Host Driver to poll clock status when changing the clock frequency. It is recommended to clear SDCLKEN bit in System Control register to remove glitch on the card clock when the frequency is changing.</p> <p>0 clock is changing frequency and not stable 1 clock is stable</p>

Table 12-11. PRSSTAT Field Descriptions (continued)

Bit	Name	Description
29	DLA	<p>Data line active. Indicates whether one of the SD_DAT line on SD bus is in use.</p> <p>For read transactions, this bit indicates if a read transfer is executing on the SD bus. Clearing this bit from 1 to 0 between data blocks generates a block gap event interrupt. This bit is set in either of the following cases:</p> <ul style="list-style-type: none"> • After the end bit of the read command • When writing a 1 to PROCTL[CREQ] to restart a read transfer <p>This bit is cleared in either of the following cases:</p> <ul style="list-style-type: none"> • When the end bit of the last data block is sent from the SD bus to the eSDHC • When beginning a read wait transfer initiated by a stop at block gap request <p>The eSDHC waits at the next block gap by driving read wait at the start of the interrupt cycle. If the read-wait signal is already driven (data buffer cannot receive data), the eSDHC can wait for current block gap by continuing to drive the read-wait signal. It is necessary to support read wait in order to use the suspend/resume function.</p> <p>For write transactions, this bit indicates that a write transfer is executing on the SD bus. Clearing this bit from 1 to 0 generates a transfer complete interrupt. This bit is set in any of the following cases:</p> <ul style="list-style-type: none"> • After the end bit of the write command • When writing a 1 to PROCTL[CREQ] to continue a write transfer <p>This bit is cleared in any of the following cases:</p> <ul style="list-style-type: none"> • When the SD card releases write-busy of the last data block, the eSDHC also detects if output is not busy. If the SD card does not drive the busy signal after CRC status is received, the eSDHC should consider the card drive not busy. • When the SD card releases write-busy prior to waiting for write transfer as a result of a stop at block gap request <p>0 SD_DAT line inactive 1 SD_DAT line active</p>
30	CDIHB	<p>Command inhibit (SD_DAT). This bit is set if the SD_DAT line is active, the read transfer active is set, or read wait is asserted. If this bit is cleared, it indicates the eSDHC can issue the next SD/MMC command. Commands with busy signal belong to command inhibit (SD_DAT) (e.g. R1b and R5b type). Clearing from 1 to 0 generates a transfer complete interrupt.</p> <p>Note: The SD host driver can save registers for a suspend transaction after this bit has cleared from 1 to 0.</p> <p>0 Can issue command which uses the SD_DAT line 1 Cannot issue command which uses the SD_DAT line</p>
31	CIHB	<p>Command inhibit (SD_CMD). This bit is cleared, if the SD_CMD line is not in use and the eSDHC can issue a SD/MMC command using the SD_CMD line.</p> <p>This bit is set immediately after the XFERTYP register is written. This bit is cleared when the command response is received. Even if the CDIHB bit is set, commands using only the SD_CMD line can be issued if this bit is cleared. Clearing from 1 to 0 generates a command complete interrupt.</p> <p>If the eSDHC cannot issue the command because of a command conflict error (refer to command CRC error) or because of command not issued by Auto CMD12 error, this bit remains set and IRQSTAT[CC] is not set. Status issuing Auto CMD12 is not read from this bit.</p> <p>0 Can issue command using only SD_CMD line 1 Cannot issue command</p>

NOTE

The host driver can issue CMD0, CMD12, CMD13 (for memory) and CMD52 (for SDIO) when the SD_DAT lines are busy during a data transfer. These commands can be issued when PRSSTAT[CIHB] is cleared. Other commands should be issued when PRSSTAT[CDIHB] is cleared. Possible changes to the *SD Physical Specification* may add other commands to this list in the future.

12.4.8 Protocol Control Register (PROCTL)

The protocol control register is shown in [Figure 12-10](#).

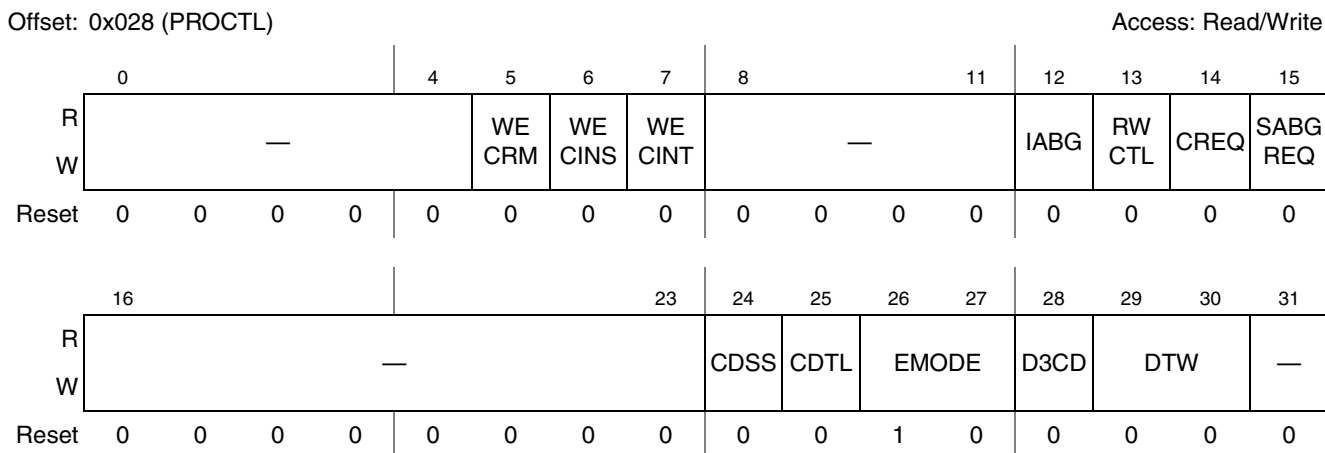


Figure 12-10. Protocol Control Register (PROCTL)

[Table 12-12](#) describes the PROCTL fields.

Table 12-12. PROCTL Field Descriptions

Bit	Name	Description
0–4	—	Reserved
5	WE CRM	Wake-up event enable on SD card removal. This bit enables wakeup event via card removal assertion in the IRQSTAT register. FN_WUS (wake-up support) in CIS does not affect this bit. 0 Disable 1 Enable
6	WE CINS	Wake-up event enable on SD card insertion. This bit enables wakeup event via card insertion assertion in the IRQSTAT register. FN_WUS (wake-up support) in CIS does not affect this bit. 0 Disable 1 Enable
7	WE CINT	Wake-up event enable on card interrupt. This bit enables wakeup event via card interrupt assertion in the IRQSTAT register. This bit can be set to 1 if FN_WUS (wake-up support) in CIS is set to 1. 0 Disable 1 Enable
8–11	—	Reserved

Table 12-12. PROCTL Field Descriptions (continued)

Bit	Name	Description
12	IABG	Interrupt at block gap. This bit is valid only in 4-bit mode of the SDIO card and selects a sample point in the interrupt cycle. If the SDIO card cannot signal an interrupt during a multiple block transfer, this bit should be cleared to avoid an inadvertent interrupt. When the host driver detects an SDIO card insertion, it should set this bit according to the CCCR of the card. 0 Disable interrupt detection during a multiple block transfer. 1 Enable interrupt detection at the block gap for a multiple block transfer.
13	RWCTL	Read wait control. The read wait function is optional for SDIO cards. If the card supports read wait, set this bit to enable the read wait protocol to stop read data using the SD_DAT[2] line. Otherwise, the eSDHC has to stop the SD clock to hold read data, which restricts command generation. When the host driver detects an SDIO card insertion, it should set this bit according to the CCCR of the card. If the card does not support read wait, this bit should never be set otherwise an SD_DAT line conflict may occur. If this bit is cleared, a stop-at-block-gap-during-read operation is also supported, but the eSDHC stops the SD clock to pause the reading operation. 0 Disable read-wait control, and stop SD clock at block gap when the SABGREQ bit is set 1 Enable read-wait control, and assert read wait without stopping the SD clock at block gap when PROCTL[SABGREQ] is set
14	CREQ	Continue request. Restarts a transaction which was stopped using the stop-at-block-gap request. To cancel the request, clear SABGREQ and set this bit to restart the transfer. The eSDHC automatically clears this bit in either of the following cases: <ul style="list-style-type: none"> For a read transaction, the PRSSTAT[DLA] bit changes from 0 to 1 as a read transaction restarts. For a write transaction, the PRSSTAT[WTA] bit changes from 0 to 1 as the write transaction restarts. Therefore, it is not necessary for the host driver to clear. If SABGREQ is set, writes to CREQ would be ignored. 0 No effect 1 Restart
15	SABGREQ	Stop at block gap request. Stops executing a transaction at the next block gap for both DMA and non-DMA transfers. Until the TC bit is set, indicating a transfer completion, the host driver should leave this bit set. Clearing SABGREQ and CREQ does not cause the transaction to restart. Read wait is used to stop the read transaction at the block gap. The eSDHC honors stop-at-block-gap request for write transfers. But for read transfers it requires that the SDIO card support read wait. Therefore, the host driver should not set this bit during read transfers unless the SDIO card supports read wait and has set read wait control to 1. Otherwise, the eSDHC stops the SD bus clock to pause the read operation during the block gap. For write transfers in which the host driver writes data to the data port register, the host driver should set this bit after all block data is written. If this bit is set, the host driver should not write data to the DATPORT register after a block is sent. When this bit is set, the host driver should not clear this bit before IRQSTAT[TC] is set. Otherwise, the eSDHC behavior is undefined. Confirm that IRQSTAT[TC] is enabled. This bit affects PRSSTAT[RTA, WTA, DLA, CIHB]. 0 Transfer 1 Stop or not resume yet
16–23	—	Reserved
24	CDSS	Card detect signal selection. Selects the source for card detection. If CDSS = 0 and D3CD = 0, then SD_CD is used. If CDSS = 0 and D3CD = 1 then DAT3 will be used for card detect 0 Card detection level is selected (for normal purpose) 1 Card detection test level is selected (for test purpose)

Table 12-12. PROCTL Field Descriptions (continued)

Bit	Name	Description
25	CDTL	Card detect test level. Determines card insertion status when CDSS is set. 0 No card in the slot 1 Card is inserted
26–27	EMODE	Endian mode. eSDHC supports only address-invariant mode in data transfer. 00 Reserved 01 Reserved 10 Address-invariant mode. Each byte location in the main memory is mapped to the same byte location in the MMC/SD card. 11 Reserved
28	D3CD	SD_DAT3 as card detection pin. If this bit is set, SD_DAT3 should be pulled down to act as a card detection pin. Be cautious when using this feature, because SD_DAT3 is chip-select for SPI mode, and a pull-down on this pin and CMD0 may set the card into SPI mode, which the eSDHC does not support. 0 SD_DAT3 does not monitor card insertion 1 SD_DAT3 is card detection pin
29–30	DTW	Data transfer width. Selects the data width of the SD bus. The host driver should set it to match the data width of the card. 00 1-bit mode 01 4-bit mode 10 Reserved 11 Reserved
31	—	Reserved

There are three ways to restart the transfer after a stop at the block gap. The appropriate method depends on whether the eSDHC issues a suspend command or the SD card accepts the suspend command:

- If the host driver does not issue a suspend command, the continue request should be used to restart the transfer.
- If the host driver issues a suspend command and the SD card accepts it, a resume command should be used to restart the transfer.
- If the host driver issues a suspend command and the SD card does not accept it, PROCTL[CREQ] should be used to restart the transfer.

Any time PROCTL[SABGREQ] stops the data transfer, the host driver should wait for IRQSTAT[TC] before attempting to restart the transfer. When restarting the data transfer by continue request, the host driver should clear PROCTL[SABGREQ] before or simultaneously.

12.4.9 System Control Register (SYSCTL)

The system control register is shown in [Figure 12-11](#).

Offset: 0x02C (SYSCTL)

Access: Mixed

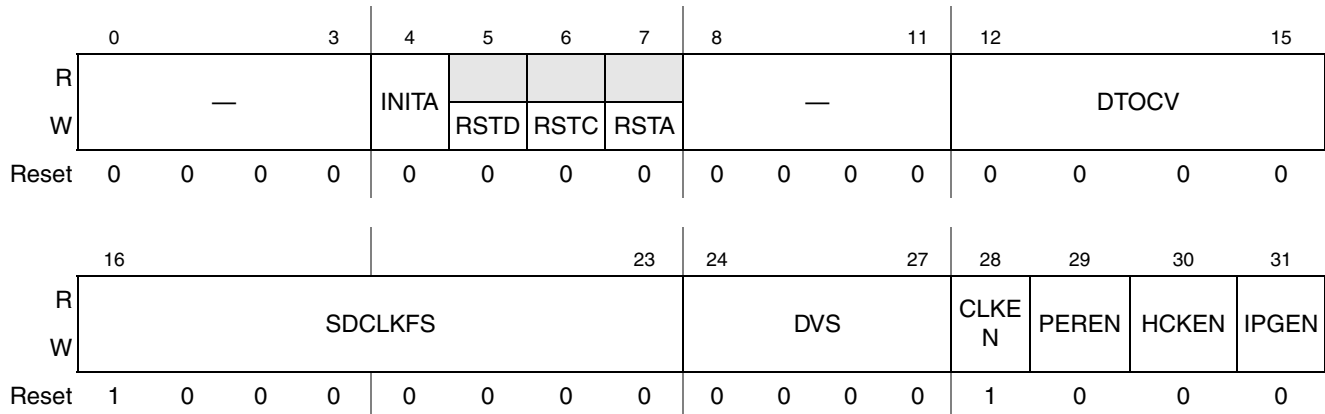


Figure 12-11. System Control Register (SYSCTL)

[Table 12-13](#) describes the SYSCTL fields.

Table 12-13. SYSCTL Field Descriptions

Bit	Name	Description
0–3	—	Reserved
4	INITA	Initialization active. When this bit is written '1', 80 SD clocks are sent to the card. After the 80 clocks are sent, this bit is self-cleared. This bit is very useful during the card power-up period when 74 SD clocks are needed and clock auto-gating feature is enabled. Writing one to this bit when it is already set has no effect. Clearing this bit at any time does not affect it. When PRSSTAT[CIHB] or PRSSTAT[CDIHB] is set, writing a one to this bit is ignored. That is, when the command line or data line is active, writing to this bit is not allowed.
5	RSTD	Software reset for SD_DAT line. The DMA and part of the data circuit are reset. The following registers and bits are cleared by this bit: <ul style="list-style-type: none"> DATPORT register Buffer is cleared and initialized; PRSSTAT register PRSSTAT[BREN, BWEN, RTA, WTA, DLA, CDIHB] PROCTL[CREQ, SABGREQ] IRQSTAT[BRR, BWR, DINT, BGE, TC] 0 Work 1 Reset
6	RSTC	Software reset for SD_CMD line. Only part of the command circuit is reset. The following bits are cleared by this bit: <ul style="list-style-type: none"> PRSSTAT[CIHB] IRQSTAT[CC] 0 Work 1 Reset

Table 12-13. SYSCTL Field Descriptions (continued)

Bit	Name	Description
7	RSTA	<p>Software reset for all. This reset affects the entire host controller except for the card-detection circuit. Register bits of type Read only, Read/Write, Read only: write-1-to-clear, and Read/Write Automatic clear are cleared.</p> <p>During its initialization, the host driver should set this bit to reset the eSDHC. The eSDHC should clear this bit when capabilities registers are valid and the host driver can read them. Additional use of this bit does not affect the value of the capabilities registers. After this bit is set, it is recommended the host driver reset the external card and re-initialize it.</p> <p>0 Work 1 Reset</p>
8–11	—	Reserved
12–15	DTOCV	<p>Data timeout counter value. Determines the interval by which SD_DAT line timeouts are detected. Refer to the data timeout error Section 12.4.10, “Interrupt Status Register (IRQSTAT)”, for information on factors that dictate timeout generation. Timeout clock frequency is generated by dividing the base clock SD_CLK value by this value. When setting this register, prevent inadvertent timeout events by clearing IRQSTATEN[DTOESEN].</p> <p>0000 SD_CLK x 2¹³ 0001 SD_CLK x 2¹⁴ ... 1110 SD_CLK x 2²⁷ 1111 Reserved</p>
16–23	SDCLKFS	<p>SD_CLK frequency select. This field, together with DVS, selects the frequency of SD_CLK pin. This bit holds the prescaler of the base clock frequency. Only the following settings are allowed:</p> <p>0x01 Base clock divided by 2 0x02 Base clock divided by 4 0x04 Base clock divided by 8 0x08 Base clock divided by 16 0x10 Base clock divided by 32 0x20 Base clock divided by 64 0x40 Base clock divided by 128 0x80 Base clock divided by 256</p> <p>Multiple bits must not be set or the behavior of this prescaler is undefined.</p> <p>According to the <i>SD Physical Specification</i> and the <i>SDIO Card Specification version 1.2 2.0</i>, the maximum SD clock frequency is 50 MHz, and should never exceed this limit. The frequency of SD_CLK is set by the following formula:</p> $\text{clock frequency} = (\text{base clock}) / [(\text{SDCLKFS} \times 2) \times (\text{DVS} + 1)] \quad \text{Eqn. 12-1}$ <p>For example, if the base clock frequency is 96 MHz, and the target frequency is 25 MHz, then choosing the prescaler value of 0x1 and divisor value of 0x1 yields 24 MHz, which is the nearest frequency less than or equal to the target. Similarly, to approach a clock value of 400 KHz, the prescaler value of 0x04 and divisor value of 0xE yields the exact clock value of 400 KHz. The reset value of this bit field is 0x80. So, if the input base clock is about 96 MHz, the default SD clock after reset is 375 KHz.</p> <p>Note: The base clock frequency equals the <code>csb_clk / SCCR[SDHCCM]</code>.</p>
24–27	DVS	<p>Divisor. Provides a more exact divisor to generate the desired SD clock frequency. The settings are as follows:</p> <p>0x0 Divide by 1 0x1 Divide by 2 ... 0xE Divide by 15 0xF Divide by 16</p>

Table 12-13. SYSCTL Field Descriptions (continued)

Bit	Name	Description
28	CLKEN	SD Card Clock Enable 0 Disable the clock 1 Enable the clock
29	PEREN	Peripheral clock enable. If set, the peripheral clock is always active and no automatic gating is applied, thus SD_CLK is active only except auto gating-off during buffer danger. If cleared, the peripheral clock is automatically off when no transaction is on the SD bus. Clearing this bit does not stop SD_CLK immediately. The peripheral clock will be internally gated off, if none of the following factors are met: <ul style="list-style-type: none"> • Command part is reset • Data part is reset • Soft reset • Command is about to send • Clock divisor is just updated • Continue request is just set • This bit is set • Card insertion is detected • Card removal is detected • Card external interrupt is detected • 80 clocks for initialization phase is ongoing 0 The peripheral clock is internally gated off 1 The peripheral clock is not automatically gated off
30	HCKEN	Master clock enable. If set, master clock is always active and no automatic gating is applied. If cleared, master clock is automatically off when no data transfer is on SD bus. Note: Master clock is the clock to the DMA engine and to the CSB interface logic. 0) Master clock is internally gated off 1) Master clock is not automatically gated off
31	IPGEN	Controller clock enable. If this bit is set, the controller clock is always active and no automatic gating is applied. The controller clock is internally gated off, if neither the following factors is met: <ul style="list-style-type: none"> • Command part is reset • Data part is reset • Soft reset • Command is about to send • Clock divisor is just updated • Continue request is just set • This bit is set • Card insertion is detected • Card removal is detected • Card external interrupt is detected • The internal bus clock is not gated off Note: The controller clock is not auto-gated off if the peripheral clock is not gated off. So, clearing this bit only does not take effect if SYSCTL[PEREN] is not cleared. 0 The controller clock is internally gated off 1 The controller clock is not automatically gated off

12.4.10 Interrupt Status Register (IRQSTAT)

An interrupt is generated when one of the status bits and its corresponding interrupt enable bit are set. For all bits, writing one to a bit clears it, while writing zero keeps the bit unchanged. More than one status can

be cleared with a single register write. For a card interrupt (IRQSTAT[CINT]), the card must stop asserting the interrupt before writing one to clear. Otherwise, the CINT bit is set again.

Figure 12-12 shows the interrupt status register.

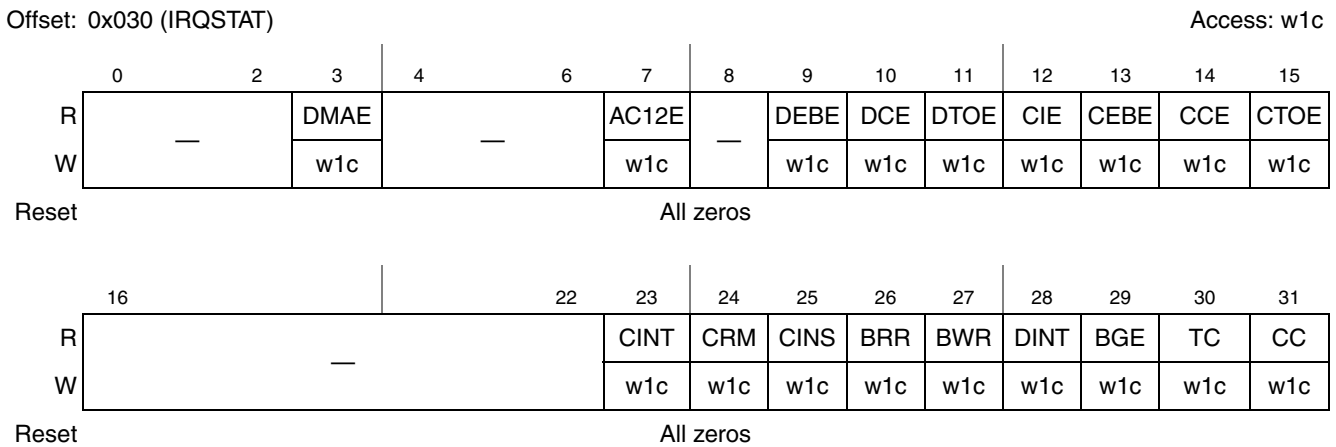


Figure 12-12. Interrupt Status Register (IRQSTAT)

Table 12-14 describes the IRQSTAT fields.

Table 12-14. IRQSTAT Field Descriptions

Bit	Name	Description
0–2	—	Reserved
3	DMAE	DMA error. Occurs when internal DMA transfer failed. This bit is set when some error occurs in the data transfer. The value in the DMA system address register is the next fetch address where the error occurs. Since any error corrupts the entire data block, the host driver should restart the transfer from the corrupted block boundary. The address of the block boundary can be calculated from the current DS_ADDR value or the remaining number of blocks and the block size. 0 No Error 1 Error
4–6	—	Reserved
7	AC12E	Auto CMD12 error. Occurs when one of the bits in AUTOC12ERR is set. This bit is also set when Auto CMD12 is not executed due to a previous command error. 0 No Error 1 Error
8	—	Reserved
9	DEBE	Data end bit error. Occurs when detecting 0 at the end bit position of read data on the SD_DAT line or at the end bit position of the CRC. 0 No Error 1 Error Note: When DEBE and CINT are set, the software should ignore DEBE. But, it must not ignore the other status bits. The software should also clear this bit by writing 1 to it. It is highly recommended to clear this bit before the next transfer.

Table 12-14. IRQSTAT Field Descriptions (continued)

Bit	Name	Description
10	DCE	Data CRC error. Occurs when detecting CRC error when transferring read data on the SD_DAT line or when detecting the write CRC status having a value other than 0b010. 0 No Error 1 Error
11	DTOE	Data timeout error. Occurs during one of following timeout conditions: <ul style="list-style-type: none"> • Busy timeout for R1b and R5b types • Busy timeout after write CRC status • Read data timeout 0 No error 1 Timeout
12	CIE	Command index error. Occurs if a command index error occurs in the command response. 0 No error 1 Error Note: Under rare conditions, CIE may be set for a command without data while a command with data is in progress. On detecting a command CRC error (IRQSTAT[CCE]) or command index error (IRQSTAT[CIE]), perform error recovery and re-issue the command without data. If Auto CMD12 is enabled for data transfer then Auto CMD12 won't be issued by hardware, so software needs to issue it after data transfer completion.
13	CEBE	Command end bit error. Occurs when the end bit of a command response is 0. 0 No error 1 End bit error generated
14	CCE	Command CRC error. A command CRC error is generated in two cases: <ul style="list-style-type: none"> • If a response is returned and IRQSTAT[CTOE] is cleared (indicating no timeout), this bit is set when detecting a CRC error in the command response. • The eSDHC detects a SD_CMD line conflict by monitoring the SD_CMD line when a command is issued. If the eSDHC drives the SD_CMD line to 1, but detects 0 on the SD_CMD line at the next SD_CLK edge, then the eSDHC aborts the command (stop driving SD_CMD line) and sets this bit. The CTOE bit is also set to distinguish the SD_CMD line conflict. 0 No error 1 CRC error generated Note: Under rare conditions, CCE may be set for a command without data while a command with data is in progress. On detecting a command CRC error (IRQSTAT[CCE]) or command index error (IRQSTAT[CIE]), perform error recovery and re-issue the command without data. If Auto CMD12 is enabled for data transfer then Auto CMD12 won't be issued by hardware, so software needs to issue it after data transfer completion.
15	CTOE	Command timeout error. Occurs if no response is returned within 64 SD_CLK cycles from the end bit of the command. Also, if eSDHC detects a SD_CMD line conflict, this bit is set along with IRQSTAT[CCE] as shown in Table 12-26 . 0 No error 1 Time out
16–22	—	Reserved

Table 12-14. IRQSTAT Field Descriptions (continued)

Bit	Name	Description
23	CINT	<p>Card interrupt.</p> <ul style="list-style-type: none"> In 1-bit mode, the eSDHC detects the card interrupt without the SD clock to support wakeup. In 4-bit mode, the card interrupt signal is sampled during the interrupt cycle. So, there are some sample delays between the interrupt signal from the SD card and the interrupt to the host system. <p>Writing 1 clears this bit. But, if the interrupt source from the SD card is not cleared, this bit is set again. To clear this bit, the SD card interrupt source must be cleared followed by writing 1 to this bit.</p> <p>When this bit is set and the host driver needs to start the interrupt service, IRQSIGEN[CINTIEN] should be cleared to stop driving the interrupt signal to the host system. After completing the card interrupt service, write 1 to clear this bit, set IRQSIGEN[CINTIEN], and start sampling the interrupt signal again.</p> <p>0 No card interrupt 1 Generate card interrupt</p>
24	CRM	<p>Card removal. This bit is set if PRSSTAT[CINS] changes from 1 to 0. When the host driver writes 1 to this bit to clear it, the status of PRSSTAT[CINS] should be confirmed. Because the card-detect state may be changed when the host driver clears this bit, an interrupt event may not be generated. When this bit is cleared, it is set again if no card is inserted. To leave it cleared, clear IRQSTATEN[CRMSEN].</p> <p>0 Card state unstable or inserted 1 Card removed</p>
25	CINS	<p>Card insertion. This bit is set if PRSSTAT[CINS] changes from 0 to 1. When the host driver writes 1 to this bit to clear it, the status of PRSSTAT[CINS] should be confirmed. Because the card-detect state may be changed when the host driver clears this bit, an interrupt event may not be generated. When this bit is cleared, it is set again if a card has been inserted. To leave it cleared, clear IRQSTATEN[CINSEN].</p> <p>0 Card state unstable or removed 1 Card inserted</p>
26	BRR	<p>Buffer read ready. This bit is set if PRSSTAT[BREN] changes from 0 to 1.</p> <p>0 Not ready to read buffer 1 Ready to read buffer</p>
27	BWR	<p>Buffer write ready. This bit is set if PRSSTAT[BWEN] changes from 0 to 1.</p> <p>0 Not ready to write buffer 1 Ready to write buffer</p>
28	DINT	<p>DMA interrupt. Occurs when the internal DMA finishes the data transfer successfully. If errors occur during data transfer, this bit is not set. Instead, the DMAE bit is set.</p> <p>0 No DMA interrupt 1 DMA interrupt is generated</p>
29	BGE	<p>Block gap event. If PROCTL[SABGREQ] is set, this bit is set when a read or write transaction is stopped at a block gap. If PROCTL[SABGREQ] is cleared, this bit is not set. During a read transaction, this bit is set at the falling edge of the SD_DAT line active status (when the transaction is stopped at SD bus timing). Read wait must be supported to use this function. During a write transaction, this bit is set at the falling edge of PRSSTAT[WTA] (after reading the CRC status at SD bus timing).</p> <p>0 No block gap event 1 Transaction stopped at block gap</p>

Table 12-14. IRQSTAT Field Descriptions (continued)

Bit	Name	Description
30	TC	<p>Transfer complete. This bit is set when a read or write transfer is completed. BLKATTR[BLKCNT] is decremented when IRQSTAT[TC] is set.</p> <p>For a read transaction, this bit is set at the falling edge of PRSSTAT[WTA]. There are two cases in which this interrupt is generated:</p> <ul style="list-style-type: none"> • When a data transfer is completed, as specified by data length (after the last data has been read to the host system). • When data has stopped at the block gap and completed the data transfer by setting PROCTL[SABGREQ] (after valid data has been read to the host system). <p>For a write transaction, this bit is set at the falling edge of PRSSTAT[DLA]. There are two cases in which this interrupt is generated:</p> <ul style="list-style-type: none"> • When the last data is written to the SD card, as specified by data length and the busy signal is released. • When data transfers are stopped at the block gap by setting PROCTL[SABGREQ] and data transfers have completed (after valid data is written to the SD card and the busy signal is released).
31	CC	<p>Command complete. This bit is set when the end bit of the command response is received (except Auto CMD12). Refer to PRSSTAT[CIHB].</p> <p>0 No command complete 1 Command complete</p>

Table 12-15 below shows that command timeout error has higher priority than command complete. If both bits are set, it can be assumed that the response was not received correctly.

Table 12-15. Relation Between Command Timeout Error and Command Complete Status

Command Complete	Command Timeout Error	Meaning of the Status
0	0	—
Don't Care	1	Response not received within 64 SD_CLK cycles
1	0	Response received

Table 12-16 below shows that transfer complete has higher priority than data timeout error. If both bits are set, the data transfer can be considered complete.

Table 12-16. Relation Between Data Timeout Error and Transfer Complete Status

Transfer Complete	Data Timeout Error	Meaning of the Status
0	0	—
0	1	Timeout occur during transfer
1	X	Data transfer complete

The relation between command CRC error and command timeout error is shown in [Table 12-17](#) below.

Table 12-17. Relation Between Command CRC Error and Command Timeout Error

Command CRC Error	Command Timeout Error	Meaning of the Status
0	0	No error
0	1	Response Timeout Error
1	0	Response CRC Error
1	1	SD_CMD line conflict

12.4.11 Interrupt Status Enable Register (IRQSTATEN)

[Figure 12-13](#) shows the interrupt status enable register. Setting the bits of IRQSTATEN enables the corresponding interrupt status bit to be set by the specified event. If any bit is cleared, the corresponding IRQSTAT bit is also cleared and is never set.

Offset: 0x034 (IRQSTATEN)

Access: Read/Write

	0	2	3	4	6	7	8	9	10	11	12	13	14	15		
R	—		DMAE SEN	—		AC12E SEN	—	DEBE SEN	DCE SEN	DTOE SEN	CIE SEN	CEBE SEN	CCE SEN	CTOE SEN		
W	—		DMAE SEN	—		AC12E SEN	—	DEBE SEN	DCE SEN	DTOE SEN	CIE SEN	CEBE SEN	CCE SEN	CTOE SEN		
Reset	0	0	0	1	0	0	0	1	0	1	1	1	1	1		
	16						22	23	24	25	26	27	28	29	30	31
R	—					CINT SEN	CRM SEN	CINS SEN	BRR SEN	BWR SEN	DINT SEN	BGE SEN	TC SEN	CC SEN		
W	—					CINT SEN	CRM SEN	CINS SEN	BRR SEN	BWR SEN	DINT SEN	BGE SEN	TC SEN	CC SEN		
Reset	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1	

Figure 12-13. Interrupt Status Enable Register (IRQSTATEN)

[Table 12-18](#) describes the IRQSTATEN fields.

Table 12-18. IRQSTATEN Field Descriptions

Bit	Name	Description
0–2	—	Reserved
3	DMAESEN	DMA error status enable 0 Masked 1 Enabled
4–6	—	Reserved
7	AC12ESEN	Auto CMD12 error status enable 0 Masked 1 Enabled
8	—	Reserved

Table 12-18. IRQSTATEN Field Descriptions (continued)

Bit	Name	Description
9	DEBESEN	Data end bit error status enable 0 Masked 1 Enabled
10	DCESSEN	Data CRC error status enable 0 Masked 1 Enabled
11	DTOESEN	Data timeout error status enable 0 Masked 1 Enabled
12	CIESEN	Command index error status enable 0 Masked 1 Enabled
13	CEBESEN	Command end bit error status enable 0 Masked 1 Enabled
14	CCESSEN	Command CRC error status enable 0 Masked 1 Enabled
15	CTOESEN	Command timeout error status enable 0 Masked 1 Enabled
16–22	—	Reserved
23	CINTSEN	Card interrupt status enable. If this bit is cleared, the eSDHC clears the interrupt request to the system. The card interrupt detection is stopped when this bit is cleared and restarted when this bit is set. To prevent inadvertent interrupts, the host driver should clear this bit before servicing the card interrupt and should set this bit again after all interrupt requests from the card are cleared. 0 Masked 1 Enabled
24	CRMSEN	Card removal status enable 0 Masked 1 Enabled
25	CINSEN	Card insertion status enable 0 Masked 1 Enabled
26	BRRSEN	Buffer read ready status enable 0 Masked 1 Enabled
27	BWRSEN	Buffer write ready status enable 0 Masked 1 Enabled
28	DINTSEN	DMA interrupt status enable 0 Masked 1 Enabled

Table 12-18. IRQSTATEN Field Descriptions (continued)

Bit	Name	Description
29	BGESEN	Block gap event status enable 0 Masked 1 Enabled
30	TCSSEN	Transfer complete status enable 0 Masked 1 Enabled
31	CCSEN	Command complete status enable 0 Masked 1 Enabled

NOTE

The eSDHC may sample the card interrupt signal during the interrupt period and hold its value in the flip-flop. As a result of synchronization, there is a delay in the card interrupt (which is asserted from the card) to the time the host system is informed.

To detect a SD_CMD line conflict, the host driver must set both CTOESEN and CCSEN bits.

12.4.12 Interrupt Signal Enable Register (IRQSIGEN)

IRQSIGEN, shown in [Figure 12-14](#), selects which interrupt status is indicated to the host system as the interrupt. These status bits all share the same interrupt line. Setting any of these bits enables an interrupt generation. The corresponding status register bit generates an interrupt when the corresponding interrupt signal enable bit is set.

Offset: 0x038 (IRQSIGEN)

Access: Read/Write

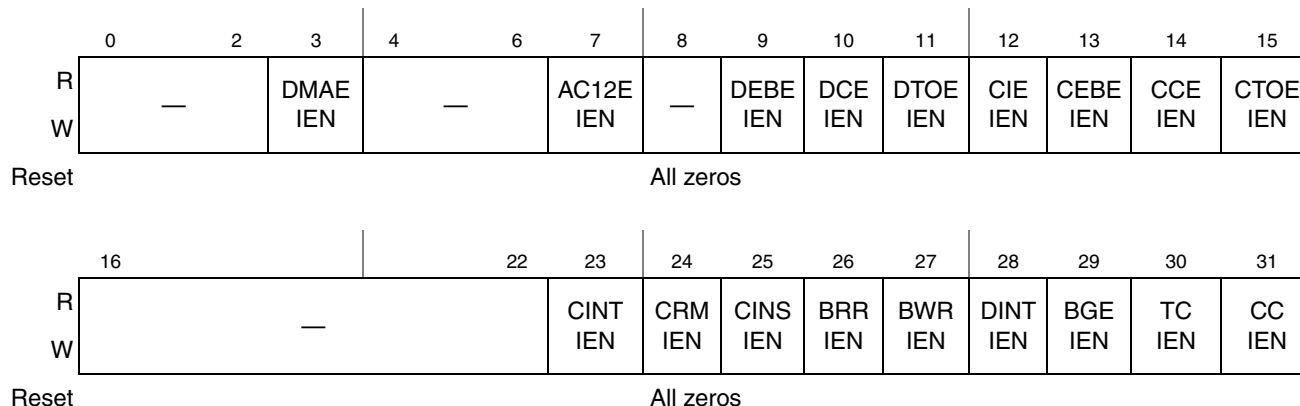

Figure 12-14. Interrupt Signal Enable Register (IRQSIGEN)

Table 12-19 describes the IRQSIGEN fields.

Table 12-19. IRQSIGEN Field Descriptions

Bit	Name	Description
0–2	—	Reserved
3	DMAEIEN	DMA error interrupt enable 0 Masked 1 Enabled
4–6	—	Reserved
7	AC12EIEN	Auto CMD12 error interrupt enable 0 Masked 1 Enabled
8	—	Reserved
9	DEBEIEN	Data end bit error interrupt enable 0 Masked 1 Enabled
10	DCEIEN	Data CRC error interrupt enable 0 Masked 1 Enabled
11	DTOEIEN	Data timeout error interrupt enable 0 Masked 1 Enabled
12	CIEIEN	Command index error interrupt enable 0 Masked 1 Enabled
13	CEBEIEN	Command end bit error interrupt enable 0 Masked 1 Enabled
14	CCEIEN	Command CRC error interrupt enable 0 Masked 1 Enabled
15	CTOEIEN	Command timeout error interrupt enable 0 Masked 1 Enabled
16–22	—	Reserved
23	CINTIEN	Card interrupt signal enable 0 Masked 1 Enabled
24	CRMIEN	Card removal interrupt enable 0 Masked 1 Enabled
25	CINSIEN	Card insertion interrupt enable 0 Masked 1 Enabled

Table 12-19. IRQSIGEN Field Descriptions (continued)

Bit	Name	Description
26	BRIEN	Buffer read ready interrupt enable 0 Masked 1 Enabled
27	BWRIEN	Buffer write ready interrupt enable 0 Masked 1 Enabled
28	DINTIEN	DMA interrupt enable 0 Masked 1 Enabled
29	BGEIEN	Block gap event interrupt enable 0 Masked 1 Enabled
30	TCIEN	Transfer complete interrupt enable 0 Masked 1 Enabled
31	CCIEN	Command complete interrupt enable 0 Masked 1 Enabled

12.4.13 Auto CMD12 Error Status Register (AUTO12ERR)

When `IRQSTAT[AC12E]` is set, the host driver checks this register to identify what kind of error Auto CMD12 indicated. This register is valid only when `IRQSTAT[AC12E]` is set.

Figure 12-15 shows the auto CMD12 error status register.

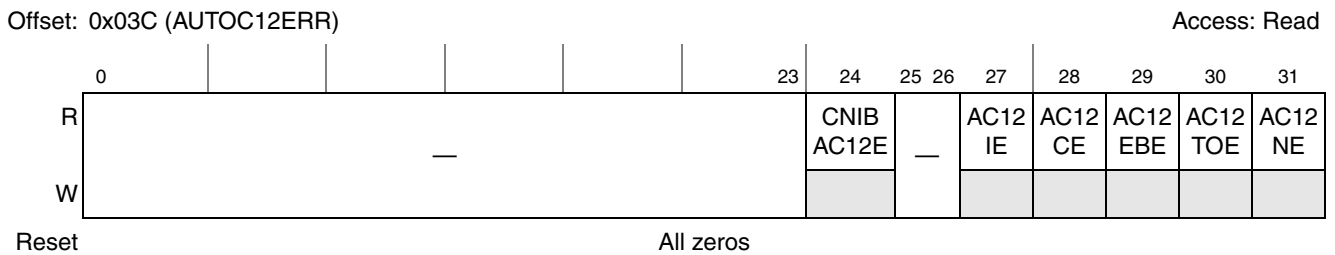

Figure 12-15. Auto CMD12 Error Status Register (AUTO12ERR)

Table 12-20 describes the IRQSTATEN fields.

Table 12-20. AUTO12ERR Field Descriptions

Bit	Name	Description
0–23	—	Reserved
24	CNIBAC12E	Command not issued by Auto CMD12 error. This bit is set when CMD_wo_DAT is not executed due to an Auto CMD12 error (D27–D30). 0 No error 1 Not Issued
25–26	—	Reserved
27	AC12IE	Auto CMD12 index error. Occurs if the command index error occurs in response to a command. 0 No error 1 Error, the CMD index in response is not CMD12
28	AC12CE	Auto CMD12 CRC error. Occurs when detecting a CRC error in the command response. 0 No CRC error 1 CRC error met in Auto CMD12 response
29	AC12EBE	Auto CMD12 end bit error. Occurs when detecting that the end bit of command response is 0 when it should be 1. 0 No error 1 End bit error generated
30	AC12TOE	Auto CMD12 timeout error. Occurs if no response is returned within 64 SD_CLK cycles from the end bit of the command. If this bit is set, the other error status bits (29–27) are meaningless. 0 No error 1 Time out
31	AC12NE	Auto CMD12 not executed. If a memory multiple block data transfer is not started due to command error, this bit is not set because it is not necessary to issue Auto CMD12. Setting this bit means eSDHC cannot issue Auto CMD12 to stop the memory multiple block data transfer due to some error. If this bit is set, the other error status bits (30–27) are meaningless. 0 Executed 1 Not executed

Table 12-21 describes the relationship between command CRC error and command timeout error for Auto CMD12.

Table 12-21. Relationship Between Command CRC Error and Command Timeout Error for Auto CMD12

Auto CMD12 CRC Error	Auto CMD12 Timeout Error	Types of Error
0	0	No error
0	1	Response timeout error
1	0	Response CRC error
1	1	SD_CMD line conflict

There are three scenarios when AUTO12ERR can be changed:

- When eSDHC is going to issue Auto CMD12
 - Set AC12NE if Auto CMD12 cannot be issued due to an error in the previous command.
 - Clear AC12NE if Auto CMD12 is issued.
- At the end bit of an Auto CMD12 response
 - Check received responses by checking the error bits 30–27.
 - Set if error is detected.
 - Clear if error is not detected.
- Before reading AUTO12ERR[CNIBAC12E]
 - Set CNIBAC12E if there is a command that cannot be issued
 - Clear CNIBAC12E if there is no command to issue

The timing of generating the Auto CMD12 error and writing to the command register is asynchronous. The command may be blocked by any Auto CMD12 error causing CNIBAC12E to be set. Therefore, it is suggested to read this register only when IRQSTAT[AC12E] is set. An Auto CMD12 error interrupt is generated when one of the error bits 31–27 is set to 1. The CNIBAC12E error bit does not generate an interrupt.

12.4.14 Host Controller Capabilities (HOSTCAPBLT)

The host controller capabilities provide the host driver with information specific to the eSDHC implementation. The value in this register does not change in software reset, and any write to this register is ignored.

Figure 12-16 shows the auto CMD12 error status register.

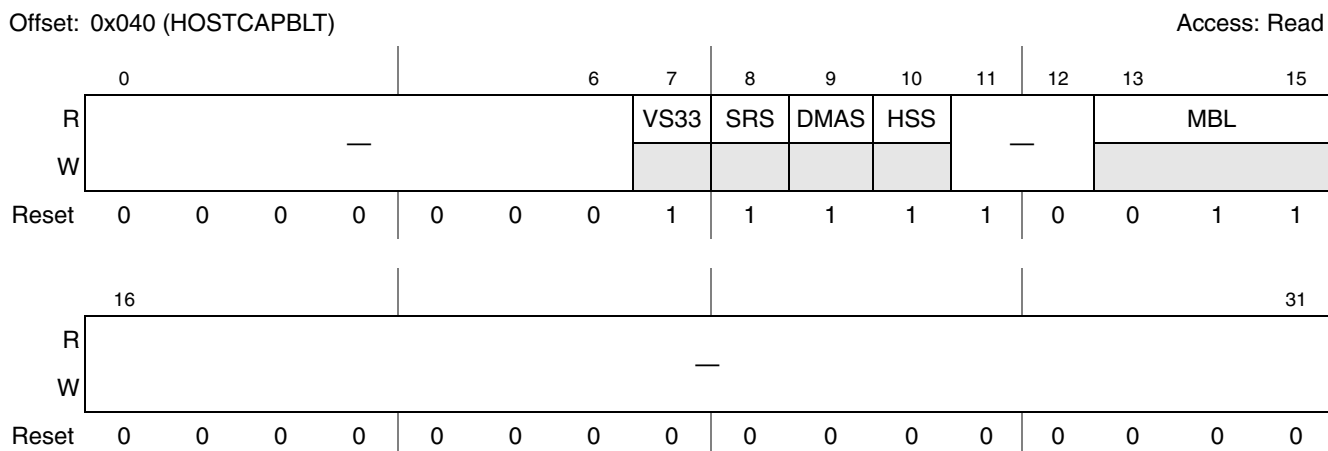


Figure 12-16. Host Capabilities Register (HOSTCAPBLT)

Table 12-22 describes the HOSTCAPBLT fields.

Table 12-22. HOSTCAPBLT Field Descriptions

Bit	Name	Description
0–6	—	Reserved
7	VS33	Voltage support 3.3 V. This bit depends on the host system ability. 0 3.3 V not supported 1 3.3 V supported Note: This is always set to 1.
8	SRS	Suspend/resume support. Indicates if eSDHC supports suspend/resume functionality. If this bit is 0, suspend and resume mechanism, as well as the read wait, are not supported and the host driver should not issue suspend or resume commands. 0 Not supported 1 Supported
9	DMAS	DMA support. Indicates if eSDHC is capable of using internal DMA to transfer data between system memory and the data buffer directly. 0 DMA not supported 1 DMA supported
10	HSS	High speed support. Indicates if the eSDHC supports high speed mode and the host system can supply the SD clock frequency from 25 to 33.25 MHz. 0 High speed supported 1 High speed supported
11–12	—	Reserved. Set to 10.
13–15	MBL	Max block length. Indicates the maximum block size that the host driver can read and write to the buffer in the eSDHC. The buffer should transfer block size without wait cycles. 000 512 bytes 001 1024 bytes 010 2048 bytes 011 4096 bytes
16–31	—	Reserved

12.4.15 Watermark Level Register (WML)

Figure 12-17 shows the watermark level register. Both write and read watermark levels are configurable. The value can be any number from 1–127 words.

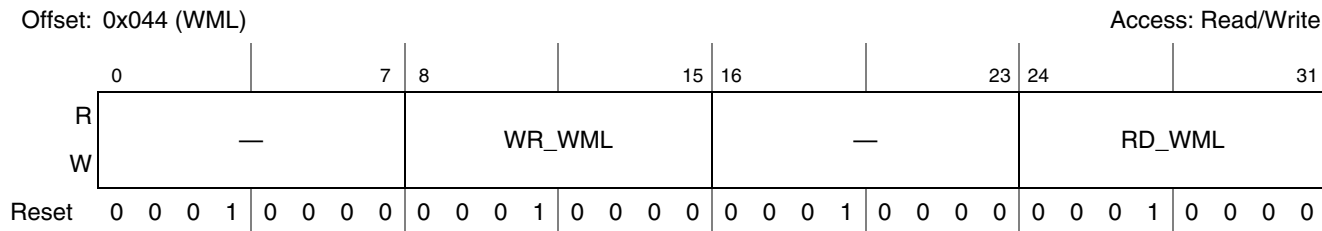


Figure 12-17. Watermark Level Register (WML)

Table 12-23 describes the WML fields.

Table 12-23. WML Field Descriptions

Bit	Name	Description
0–7	—	Reserved.
8–15	WR_WML	Write watermark level. Number of 32-bit words of watermark level in write data transfer. Note: The minimum value is 0x02, which represents 2 words (8 bytes).
16–23	—	Reserved.
24–31	RD_WML	Read watermark level. Number of 32-bit words of watermark level in read data transfer.

12.4.16 Force Event Register (FEVT)

The force event register is not a physically implemented register. Rather, it is an address to which the IRQSTAT register can be written if the corresponding bit of IRQSTATEN is set. Therefore, this register is a write-only register and writing zero has no effect. Writing 1 to this register sets the corresponding bit of IRQSTAT. Reading from this register always returns zeroes.

Forcing a card interrupt generates a short pulse on the SD_DAT[1] line, and the driver may treat this interrupt as normal. The interrupt service routine may skip polling the card-interrupt source as the interrupt is self-cleared.

Figure 12-18 shows the force event register.

Offset: 0x050 (FEVT)

Access: Write

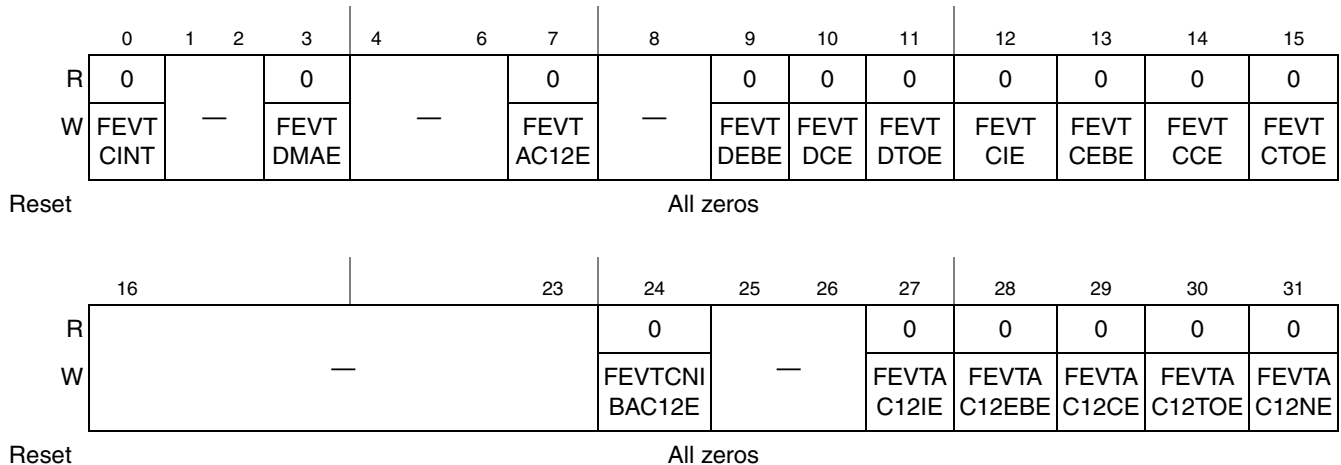


Figure 12-18. Force Event Register (FEVT)

Table 12-24 describes the FEVT fields.

Table 12-24. FEVT Field Descriptions

Bit	Name	Description
0	FEVTCINT	Force event card interrupt. Writing 1 to this bit generates a low-level short pulse on the internal SD_DAT[1] line, which imitates a self-clearing interrupt from the external card. If enabled, IRQSTAT[CINT] is set and the interrupt service routine may treat this interrupt as a normal interrupt from the external card.
1–2	—	Reserved
3	FEVTDMAE	Force event DMA error. Forces IRQSTAT[DMAE] to set.
4–6	—	Reserved
7	FEVTAC12E	Force event Auto CMD12 error. Forces IRQSTAT[AC12E] to set.
8	—	Reserved
9	FEVTDEBE	Force event data end bit error. Forces IRQSTAT[DEBE] to set.
10	FEVTDCE	Force event data CRC error. Forces IRQSTAT[DCE] to set.
11	FEVTDTOE	Force event data time out error. Forces IRQSTAT[DTOE] to set.
12	FEVTCIE	Force event command index error. Forces IRQSTAT[CCE] to set.
13	FEVTCEBE	Force event command end bit error. Forces IRQSTAT[CEBE] to set.
14	FEVTCCE	Force event command CRC error. Forces IRQSTAT[CCE] to set.
15	FEVCTOE	Force event command time out error. Forces IRQSTAT[CTOE] to set.
16–23	—	Reserved
24	FEVTCNIBAC12E	Force event command not executed by Auto CMD12 error. Forces AUTOC12ERR[CNIBAC12E] to set.
25–26	—	Reserved
27	FEVTAC12IE	Force event Auto CMD12 index error. Forces AUTOC12ERR[AC12IE] to set.
28	FEVTAC12EBE	Force event Auto CMD12 end bit error. Forces AUTOC12ERR[AC12EBE] to set.
29	FEVTAC12CE	Force event Auto CMD12 CRC error. Forces AUTOC12ERR[AC12CE] to set.
30	FEVTAC12TOE	Force event Auto CMD12 time out error. Forces AUTOC12ERR[AC12TOE] to set.
31	FEVTAC12NE	Force event Auto CMD12 not executed. Forces AUTOC12ERR[AC12NE] to set.

12.4.17 Host Controller Version Register (HOSTVER)

The host controller version register, shown in [Figure 12-19](#), contains the version for the vendor and the host controller. All the bits are read-only.

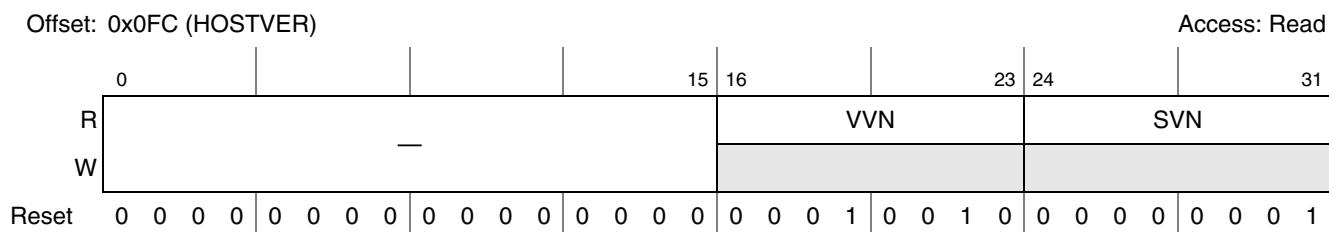


Figure 12-19. Host Controller Version Register (HOSTVER)

[Table 12-25](#) describes the HOSTVER fields.

Table 12-25. HOSTVER Field Descriptions

Bit	Name	Description
0–15	—	Reserved
16–23	VVN	Vendor version number. The host driver should not use this status. The upper and the lower 4-bits indicate the version. 0x12 Freescale eSDHC <i>version 2.2</i> others Reserved
24–31	SVN	Specification version number. Indicates for the host controller specification version. 0x01 SD Host Specification <i>Version 2.0</i> , supports the test event register. others Reserved

12.4.18 DMA Control Register (DCR)

This is implemented as SDHCCR as described in [Section 6.3.2.11](#).

12.5 Functional Description

The following sections provide a brief functional description of the major system blocks, including the data buffer, DMA CSB interface, register bank, register bus interface, dual-port memory wrapper, data/command controller, clock and reset manager, and clock generator.

12.5.1 Data Buffer

The eSDHC uses one configurable data buffer so that data can be transferred between the internal system bus (register bus or CSB bus) and the SD card in an optimized manner to maximize throughput between the two clock domains (the IP peripheral clock and the master clock). See [Figure 12-20](#) for an illustration of the buffer scheme.

The buffer is used as temporary storage for data being transferred between the host system and the card. The water mark levels for read and write are both configurable and can be any value between 1 and 127 words.

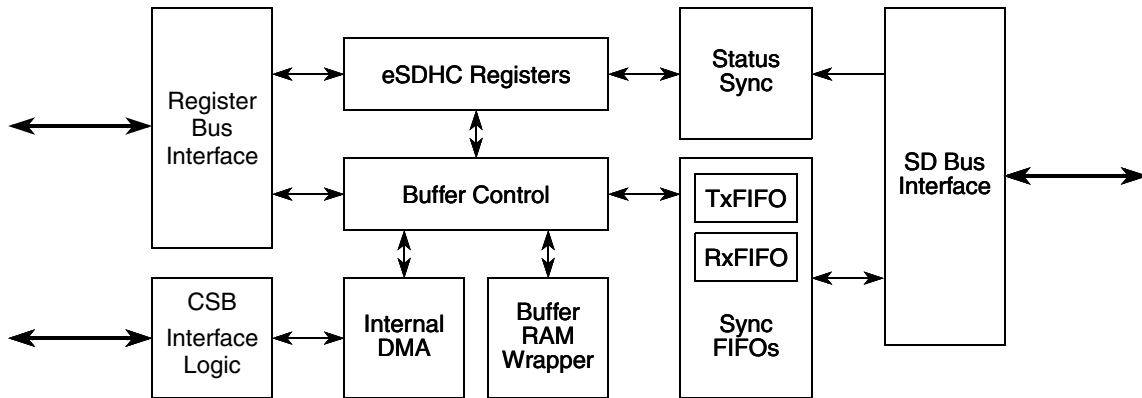


Figure 12-20. eSDHC Buffer Scheme

For a host read operation, when the amount of data exceeds the RD_WML value, the eSDHC sets PRSSTAT[BREN] and either:

- Issues a DMA request to inform the system to read the data
- Issues a DMA interrupt to inform the system to read the data

When granted CSB access permission, the internal DMA burst-reads RD_WML number of words

Conversely, for a host write operation, when the amount of buffer spaces exceeds the WR_WML value, the eSDHC sets PRSSTAT[BWEN] and either:

- Issues a DMA request to inform the system to write data to the buffer
- Issues a DMA interrupt to inform the system to write data to the buffer

When granted CSB access permission, the internal DMA burst-writes WR_WML number of words into the buffer

12.5.1.1 Write Operation Sequence

There are two ways to write data into the buffer when the user transfers data to the card:

- Processor core polling IRQSTAT[BWR] (interrupt or polling)
- Internal DMA

When the internal DMA is not used (XFERTYP[DMAEN] is not set when the command is sent), and more than WML[WR_WML] number of empty word slots are available and ready for receiving new data, the eSDHC sets the IRQSTAT[BWR]. The buffer write ready interrupt is generated if it is enabled by software.

When the internal DMA is used, the eSDHC does not inform the system before all the required number of bytes are transferred and no error is encountered. When an error occurs during the data transfer, the eSDHC aborts the data transfer and abandons the current block. The host driver should read the content of the DMA system address register to obtain the start address of abandoned data block. If the current data transfer is in multi-block mode, the eSDHC does not automatically send CMD12 even though XFERTYP[AC12EN] is set. Therefore, in this scenario, the host driver should send CMD12 and restart the write operation from that address. It is recommended that software reset for data is applied before the transfer is restarted after error recovery.

The eSDHC does not start fetching the data from the host system, until the WML[WR_WML] number of words of data can be held in the buffer. If the buffer is empty and the host system does not write data in time, the eSDHC stops the SD_CLK to avoid a data buffer underrun situation.

12.5.1.2 Read Operation Sequence

There are two ways to read data from the buffer when transferring data from the card:

- Processor core polling IRQSTAT[BRR] (interrupt or polling)
- Internal DMA

When the internal DMA is not used (XFERTYP[DMAEN] is not set when the command is sent), and more than WML[RD_WML] number of words are available and ready for the system to fetch data, the eSDHC sets the IRQSTAT[BRR]. The buffer read ready interrupt is generated if it is enabled by software.

When the internal DMA is used, the eSDHC does not inform the system before all the required number of bytes are transferred and no error is encountered. When an error occurs during the data transfer, the eSDHC aborts the data transfer and abandons the current block. The host driver should read the content of the DMA system address register to obtain the start address of abandoned data block. If the current data transfer is in multi-block mode, the eSDHC does not automatically send CMD12 even though XFERTYP[AC12EN] is set. Therefore, in this scenario, the host driver should send CMD12 and restart the read operation from that address. It is recommended that software reset for data is applied before the transfer is restarted after error recovery.

The eSDHC does not start data transmission until the WML[RD_WML] number of words of data are in the buffer. If the buffer is full and the host system does not read the data in time, the eSDHC stops the SD_CLK to avoid a data buffer overrun situation.

12.5.1.3 Data Buffer Size

To use the buffer in the most optimized way, the buffer size must be known. In the eSDHC the data buffer can hold up to 128 32-bit words, and the read and write watermark levels are each configurable from 1–128 words. The host driver may configure the values according to the system situation and requirements.

During multi-block data transfer, the maximum block length is 4096 bytes, which can satisfy all the requirements from MMC, SDIO, and SD cards. Any block length less than this value is also allowed. The only restriction is from the external card since it may not support such a large block or a partial block access that is not an integer multiple of 512 bytes.

12.5.1.4 Dividing Large Data Transfer

This SDIO command CMD53 definition limits the maximum data size of data transfers according to the following formula:

$$\text{Maximum data size} = (\text{block size}) \times (\text{block count}) \quad \text{Eqn. 12-2}$$

The length of a multiple block transfer must be in block size units. If the total data length cannot be divided evenly to a multiple of the block size, then there are two ways to transfer the data depending on the function and card design.

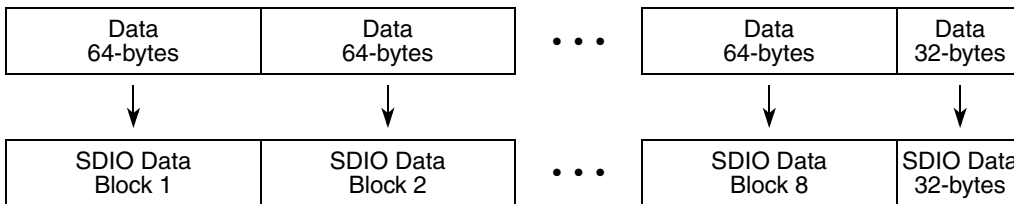
- The card driver splits the transaction. The remainder of block size data is then transferred using a single block command at the end.
- Add dummy data in the last block to fill the block size. The card must remove the dummy data.

See [Figure 12-21](#) for an example of dividing large data transfers. Although the eSDHC supports a block size of up to 4096 bytes, the example below illustrates a maximum of 64 bytes where the data must be divided.

544-bytes WLAN Frame



WLAN Frame is divided equally into 64-byte blocks plus the remainder 32-bytes



Eight 64-byte blocks are sent in Block Transfer Mode and the remainder 32-bytes are sent in Byte Transfer Mode

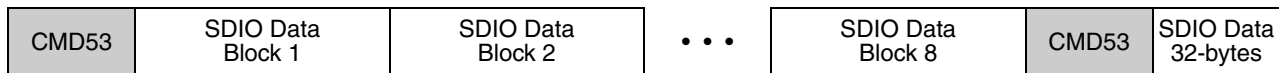


Figure 12-21. Example of Dividing a Large Data Transfer

12.5.2 DMA CSB Interface

The internal DMA implements a DMA engine and CSB master. When the internal DMA is enabled (XFERTYP[DMAEN] is set), the buffer interrupt status bits are still set if they are enabled. To avoid setting them, clear IRQSTATEN[BWRSEN, BRRSEN]. See [Figure 12-22](#) for illustration of the DMA CSB interface block. The internal DMA must not be used to read (or write) data if the data will be written (or read) by the CPU through the DATPORT register.

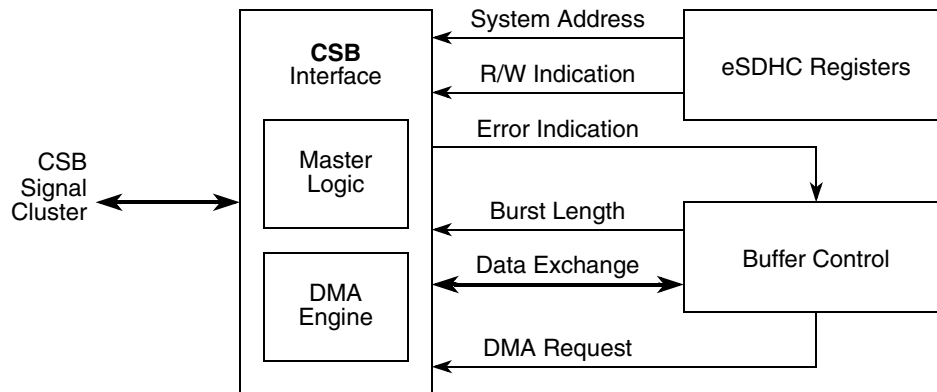


Figure 12-22. DMA CSB Interface Block

12.5.2.1 Internal DMA Request

If the watermark level is met in the data transfer and the internal DMA is enabled, the data buffer block sends a DMA request to DMA engine. The delay of response from the internal DMA engine depends on the system bus loading and the priority assigned to eSDHC. The DMA engine does not respond to the request during its burst transfer, and is available as soon as the burst is over. The data buffer deasserts the request once an access on the buffer is made. Upon access to the buffer by the internal DMA, the data buffer updates its internal buffer pointer and when the watermark level is satisfied, another DMA request is sent.

12.5.2.2 DMA Burst Length

DMA burst length is set to a default value to transfer the data into and out of the system bus. This value is not programmable by the user.

12.5.2.3 CSB Master Interface

It is possible that the internal DMA engine fails during the data transfer. When an error occurs, the DMA engine stops the transfer and goes to the idle state, while the internal data buffer stops working, too. `IRQSTAT[DMAE]` is set to inform the driver.

Once the `IRQSTAT[DMAE]` interrupt is received, software should send `CMD12` to abort the current transfer and read `DSADDR[DS_ADDR]` to obtain the start address of the corrupted block. After the DMA error is fixed, the software should apply a data reset and restart the transfer from this address to recover the corrupted block.

12.5.3 SD Protocol Unit

The SD protocol unit deals with all SD protocol affairs and performs the following:

- Acts as the bridge between internal buffer and the SD bus
- Sends the command data and its argument serially
- Stores the serial response bit stream into corresponding registers

- Detects bus state on SD_DAT[0] line
- Monitors interrupt from the SDIO card
- Asserts read wait signal
- Gates off SD clock when the buffer announces danger status
- Detects write-protect state and other functions

It consists of four submodules: SD transceiver, SD clock and monitor, command agent and data agent.

12.5.3.1 SD Transceiver

In the SD protocol unit, the transceiver is the main control module. It consists of a FSM and the control module, from which the control signals for all other three modules are generated.

12.5.3.2 SD Clock and Monitor

This module monitors the signal level on all four data lines and the command lines, directly route the level values into the register bank for the driver to debug with.

The transceiver reports the card insertion state according to the $\overline{\text{SD_CD}}$ state, or signal level on SD_DAT[3] line when PROCTL[D3CD] is set.

The module detects the SD_WP (write protect) line. With the information of SD_WP state, the register bank ignores the command accompanied by write operation, when the SD_WP switch is on.

If the internal data buffer is in danger and the SD clock must be gated off to avoid buffer over/underrun, this module asserts the gate of output SD clock to shut the clock off. When the buffer danger is eliminated when system access of the buffer catches up, the clock gate of this module is open and the SD clock is active again.

12.5.3.3 Command Agent

The command agent deals with the transactions on SD_CMD line. See [Figure 12-23](#) for illustration of the structure for the command CRC shift register.

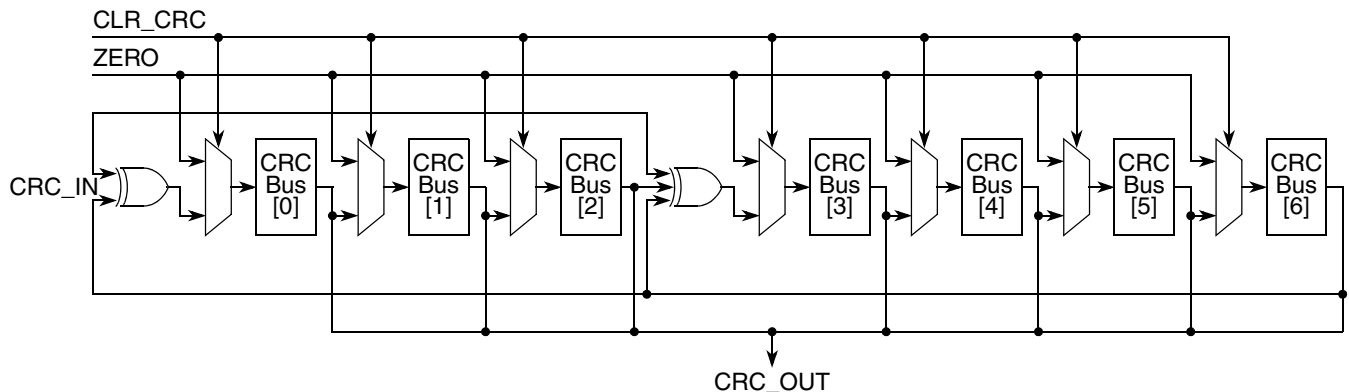


Figure 12-23. Command CRC Shift Register

The CRC polynomials for the SD_CMD are as follows:

Generator polynomial: $G(x) = x^7 + x^3 + 1$

$M(x) = (\text{first bit}) \times x^n + (\text{second bit}) \times x^{n-1} + \dots + (\text{last bit}) \times x^0$

$\text{CRC}[6:0] = \text{Remainder} [(M(x) \times x^7) \div G(x)]$

12.5.3.4 Data Agent

The data agent handles the transactions on the four data lines. Moreover, this module also detects the busy state from on SD_DAT[0] line, and generates read wait state by the request from the transceiver. The CRC polynomials for the SD_DAT are as follows:

Generator polynomial: $G(x) = x^{16} + x^{12} + x^5 + 1$

$M(x) = (\text{first bit}) \times x^n + (\text{second bit}) \times x^{n-1} + \dots + (\text{last bit}) \times x^0$

$\text{CRC}[15:0] = \text{Remainder} [(M(x) \times x^{16}) \div G(x)]$

12.5.4 Clock & Reset Manager

This module controls all the reset signals within the eSDHC. There are four types of reset signals within eSDHC: hardware reset, software reset for all, software reset for data, and software reset for command. All these signals are fed into this module and stable signals are generated inside the module to reset all other modules.

This module also gates off all the inside signals. The module monitors the activities of all other modules, supplies the clocks for them, and when enabled, automatically gates off the corresponding clocks.

12.5.5 Clock Generator

The clock generator generates the SD_CLK by dividing the internal bus clock into two stages. Refer to [Figure 12-24](#) for the structure of the divider, in which the term base represents the frequency of the internal bus clock. Refer to SYSCTL[SDCLKFS] and SYSCTL[DVS] (see [Section 12.4.9, “System Control Register \(SYSCTL\)”](#)) to select the divisor values.

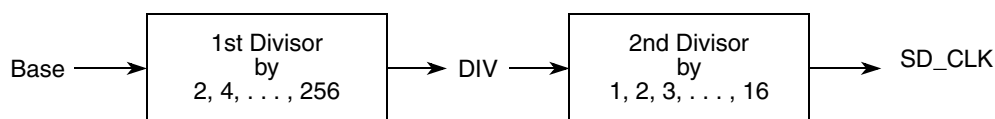


Figure 12-24. Two Stages of Clock Divider

The first stage is a prescaler. The frequency of clock output from this stage, DIV, can be base/2, base/4, ..., or base/256.

The second stage outputs the actual clock, SD_CLK, as the driving clock for all sub-modules of SD protocol unit, and the sync FIFOs in [Figure 12-20](#) to synchronize with the data rate from the internal data buffer. It can be div, div/2, div/3, ..., or div/16. Thus, the highest frequency of SD_CLK generated by the internal bus clock is base while the lowest frequency is base/4096.

12.5.6 SDIO Card Interrupt

This section discusses interrupts in 1- and 4-bit modes as well as card interrupt handling.

12.5.6.1 Interrupts in 1-bit Mode

In this case, the SD_DAT[1] pin is dedicated to providing the interrupt function. An interrupt is asserted by pulling the SD_DAT[1] low from the SDIO card, until the interrupt service is finished to clear the interrupt.

12.5.6.2 Interrupt in 4-bit Mode

As the interrupt and data line 1 share pin 8 in four-bit mode, an interrupt is only sent by the card and recognized by the host during a specific time. This is known as the interrupt period. The eSDHC only samples the level on pin 8 during the interrupt period. At all other times, the host ignores the level on pin 8 and treats it as the data signal. The definition of the interrupt period is different for operations with single- and multiple-block data transfers.

For normal single data block transmissions, the interrupt period becomes active two clock cycles after the completion of a data packet. This interrupt period lasts until after the card receives the end bit of the next command that has a data block transfer associated with it.

For multiple block data transfers in 4-bit mode there is only a limited period of time that the interrupt period can be active due to the limited period of data line availability between the multiple blocks of data. This requires a more strict definition of the interrupt period. For this case, the interrupt period is limited to two clock cycles, which begins two clocks after the end bit of the previous data block. During this two-clock cycle interrupt period if an interrupt is pending, the SD_DAT[1] line is held low for one clock cycle with the last clock cycle pulling SD_DAT[1] high. On completion of the interrupt period, the card releases the SD_DAT[1] line into the high-Z state. The eSDHC samples the SD_DAT[1] during the interrupt period when PROCTL[IABG] is set.

for further information about the SDIO card interrupt, see *SDIO Card Specification v2.0*.

12.5.6.3 Card Interrupt Handling

When IRQSIGEN[CINTIEN] is cleared, the eSDHC clears the interrupt request to the host system. The host driver should clear this bit before servicing the SDIO interrupt and should set this bit again after all interrupt requests from the card are cleared to prevent inadvertent interrupts.

If enabled by IRQSTATEN[CINTSEN], the IRQSTAT[CINT] bit can only be cleared by resetting the SDIO interrupt source and then writing one to this bit. Merely writing to this bit has no effect.

In 1-bit mode, the eSDHC detects the SDIO interrupt with or without SD clock (to support wakeup). In 4-bit mode, the interrupt signal is sampled during the interrupt period, so there are some sample delays between the interrupt signal from the SDIO card and the interrupt to the host system interrupt controller. When IRQSTAT[CINT] is set and the host driver needs to start this interrupt service, IRQSTATEN[CINTSEN] is cleared in order to clear IRQSTAT[CINT] that is latched in the eSDHC and to stop driving the interrupt signal to the processor's interrupt controller. The host driver must issue a CMD52 to clear the interrupts at the card. After completion of the card interrupt service, IRQSTATEN[CINTSEN] is set, and the eSDHC can start sampling the interrupt signal again.

See the following illustrations:

- [Figure 12-25 \(a\)](#) for an illustration of the SDIO card interrupt scheme

- Figure 12-25 (b) for the sequences of software and hardware events that take place during card interrupt handling procedure

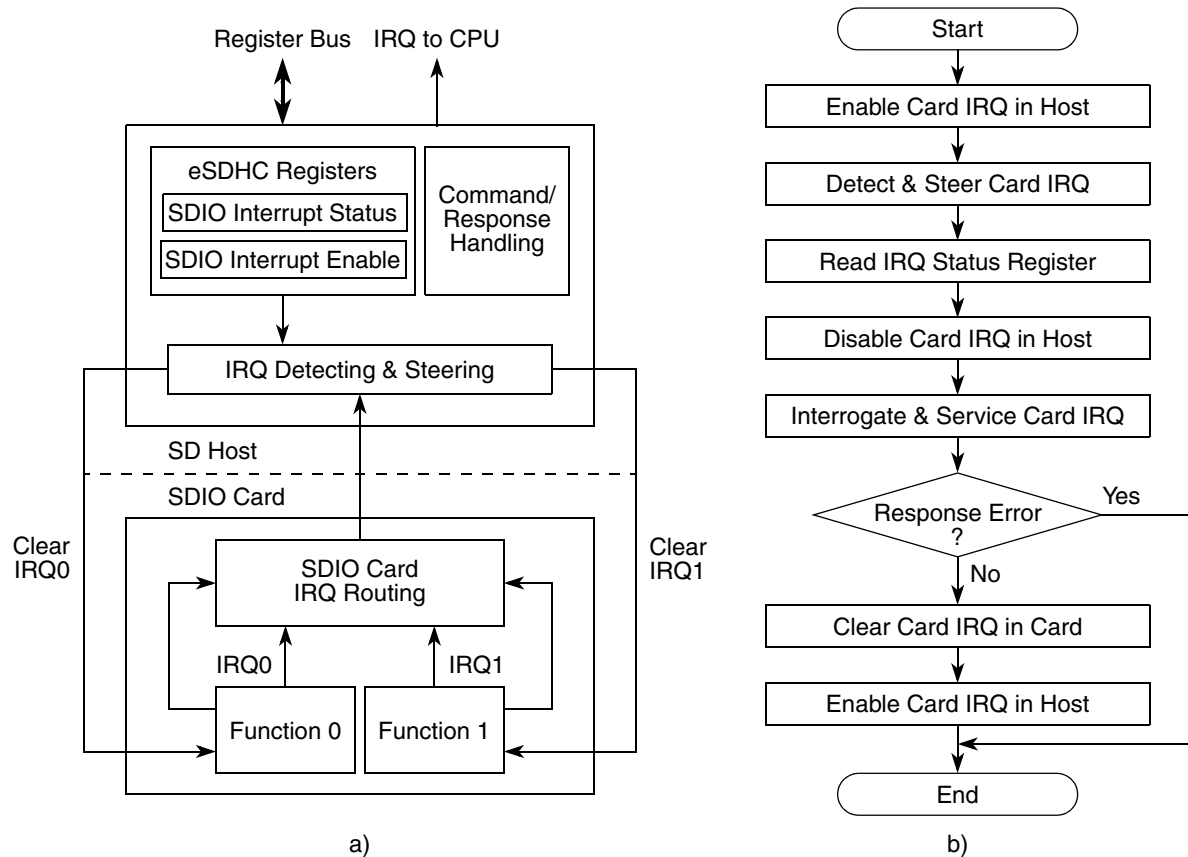


Figure 12-25. a) Card Interrupt Scheme; b) Card Interrupt Detection and Handling Procedure

12.5.7 Card Insertion and Removal Detection

The eSDHC uses the $SD_DAT[3]$ pin or the SD_CD pin to detect card insertion or removal. When $SD_DAT[3]$ pin is used for card detection, user needs to pull-down this pad as a default state. When there is no card on the MMC/SD bus, the $SD_DAT[3]$ is pulled to a low voltage level by default. When any card is inserted to or removed from the socket, the eSDHC detects the logic value changes on the $SD_DAT[3]$ pin and generates an interrupt.

When $SD_DAT[3]$ pin is not used for card detection, SD_CD must be connected for card detection. It may be implemented by a GPIO. Whether $SD_DAT[3]$ is configured for card detection or not, SD_CD is always a reference for card detection, either $SD_DAT[3]$ or SD_CD reports card inserted, the eSDHC informs the host system that a card is inserted, and the interrupt is sent if it is enabled.

12.5.8 Power Management

When there is no operation between eSDHC and the card through SD bus, the internal clocks in the chip level clock control module can be completely disabled to save power. This can be done by clearing the $SCCR[SDHCCM]$ bits.

12.6 Initialization/Application Information

All communication between system and cards are controlled by the host. The host sends commands of two types: broadcast and addressed (point-to-point) commands. Note that eSDHC supports only one SDIO card.

Broadcast commands are intended for all cards, such as GO_IDLE_STATE, SEND_OP_COND and ALL_SEND_CID. In broadcast mode, all cards are in the open-drain mode to avoid bus contention. For the commands of bc and bcr categories, see [Section 12.6.5, “Commands for MMC/SD/SDIO.”](#)

After the broadcast command CMD3 is issued, the cards enter standby mode. Addressed type commands are used from this point. In this mode, the SD_CMD/SD_DAT I/O pads turn to push-pull mode, to have the driving capability for maximum frequency operation. For the commands of ac and adtc categories, see [Section 12.6.5, “Commands for MMC/SD/SDIO.”](#)

12.6.1 Command Send and Response Receive Basic Operation

Assuming data type WORD is an unsigned 32-bit integer, the below flow is a guideline for sending a command to the card(s):

```

send_command(cmd_index, cmd_arg, other requirements)
{
WORD wCmd; // 32-bit integer to make up the data to write into the XFERTYP register, it is
           // recommended to implement in a bit-field manner
wCmd = (<cmd_index> & 0x3f) << 24; // set the first 8 bits as '00'+<cmd_index>
set CMDTYP, DPSEL, CICEEN, CCCEN, RSTTYP, and DTDSEL according to the command index;
           // XFERTYP register bits
if (internal DMA is used) wCmd |= 0x1;
if (multi-block transfer) {
    set XFERTYP[MSBSEL] bit;
    if (finite block number) {
        set XFERTYP[BCEN] bit;
        if (autol2 command is to use) set XFERTYP[AC12EN] bit;
    }
}
write_reg(CMDARG, <cmd_arg>); // configure the command argument
write_reg(XFERTYP, wCmd); // set XFERTYP register as wCmd value to issue the command
}
wait_for_response(cmd_index)
{
while (IRQSTAT[CC] is not set); // wait until command complete bit is set
read IRQSTAT register and check if any error bits about command are set;
if (any error bits are set) report error;
write 1 to clear IRQSTAT[CC] and all command error bits;
}
    
```

For the sake of simplicity, the function wait_for_response is implemented here by means of polling. For an effective and formal way, the response is usually checked after the command complete interrupt is received. By doing this, ensure the corresponding interrupt status bits are enabled.

For some scenarios, the response timeout is expected. For instance, after all cards respond to CMD3 and go to the standby state, no response to the host when CMD2 is sent. The host driver should manage false errors similar to this with caution.

12.6.2 Card Identification Mode

When a card is inserted to the socket or the card was reset by the host, the host needs to validate the operation voltage range, identify the cards, and request the cards to publish the relative card address (RCA) or to set the RCA for the MMCs.

12.6.2.1 Card Detect

See [Figure 12-26](#) for a flow diagram showing the detection of MMC, SDIO, and SD cards using the eSDHC.

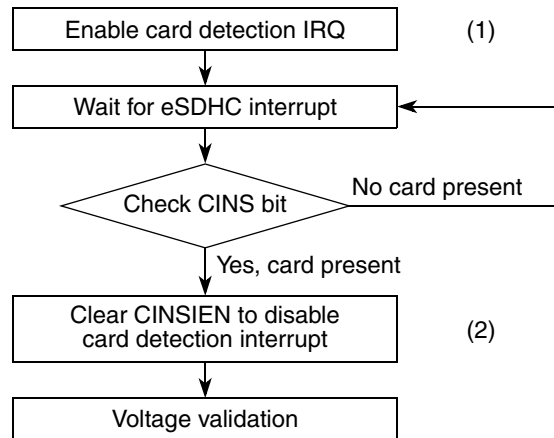


Figure 12-26. Flow Diagram for Card Detection

- Set IRQSIGEN[CINIEN] to enable card detection interrupt.
- When an interrupt from eSDHC is received, check IRQSTAT[CINS] to see if it is caused by card insertion.
- Clear the IRQSIGEN[CINIEN] to disable card detection interrupt and ignore all card insertion interrupt afterwards.

12.6.2.2 Reset

The host consists of the following three types of reset:

- Hardware reset (card and host) which is driven by POR (power on reset).
- Software reset (host only) is proceeded by the write operation on the SYSCTL[RSTD], SYSCTL[RSTC], or SYSCTL[RSTA] bits to reset the data part, command part, or all parts of the host controller, respectively.
- Card reset (card only). The command CMD0, GO_IDLE_STATE, is the software reset command for all types of MMCs and SD memory cards. This command sets each card into idle state regardless of the current card state. For an SDIO card, CMD52 is used to write I/O reset in CCCR. The cards are initialized with a default relative card address (RCA = 0x0000) and with a default driver stage register setting (lowest speed, highest driving current capability).

After the card is reset, the host needs to validate the voltage range of the card. See [Figure 12-27](#) for the software flow to reset the eSDHC and card.

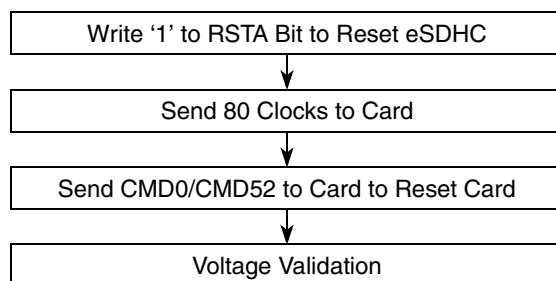


Figure 12-27. Flow Chart for Reset of eSDHC and SD I/O Card

```

software_reset()
{
    set_bit(SYSCTL, RSTA);           // software reset the host
    set SYSCTL[DTOCV and SDCLKFS];  // get the SD_CLK of frequency around 400 KHz
    poll PRSTAT[CIHB and CDIHB];    // wait until both bits are cleared
    set_bit(SYSCTRL, INTIA);        // send 80 clock ticks for card to power-up
    send_command(CMD_GO_IDLE_STATE, <other parameters>); // reset the card with CMD0
    or send_command(CMD_IO_RW_DIRECT, <other parameters>);
}
  
```

12.6.2.3 Voltage Validation

All cards should be able to establish communication with the host using any operation voltage in the maximum allowed voltage range specified in this standard. However, the supported minimum and maximum values for V_{DD} are defined in the operation conditions register (OCR) and may not cover the whole range. Cards that store the CID (card identification) and CSD data in the preloaded memory are only able to communicate this information under data transfer V_{DD} conditions. This means that if the host and card have different V_{DD} ranges, the card is not able to complete the identification cycle, nor is it able to send CSD data.

Therefore, a special command is available:

- SEND_OP_CONT (CMD1 for MMC),
- SD_SEND_OP_CONT (ACMD41 for SD Memory)
- IO_SEND_OP_CONT (CMD5 for SDIO).

The voltage validation procedure is designed to provide a mechanism to identify and reject cards which do not match the V_{DD} range(s) desired by the host. This is accomplished by the host sending the desired V_{DD} voltage window as the operand of this command. Cards that can not perform data transfer in the specified range must discontinue any further bus operations and enter the inactive state. By omitting the voltage range in the command, the host can query each card and determine the common voltage range before sending out-of-range cards into the inactive state. This query should be used if the host is able to select a common voltage range or if a notification should be sent to the system when a non-usable cards in the stack is detected.

The following steps illustrate how to perform voltage validation when a card is inserted:

```

voltage_validation(voltage_range_arguement)
{
label the card as UNKNOWN;
send_command(IO_SEND_OP_COND, 0x0, <other parameters are omitted>);
    // CMD5, check SDIO operation voltage, command argument is zero
if (RESP_TIMEOUT != wait_for_response(IO_SEND_OP_COND)) { // SDIO command is accepted
    if (0 < number of IO functions) {
        label the card as SDIO;
        IORDY = 0;
        while (!(IORDY in IO OCR response)) { // set voltage range for each IO function
            send_command(IO_SEND_OP_COND, <voltage range>, <other parameter>);
            wait_for_response(IO_SEND_OP_COND);
        } // end of while ...
    } // end of if (0 < ...)
    if (memory part is present inside SDIO card) label the card as SDCombo;
        // this is an SD-Combo card
} // end of if (RESP_TIMEOUT...)
if (the card is labeled as SDIO card) return;
    // card type is identified and voltage range is set, so exit the function;
send_command(APP_CMD, 0x0, <other parameters are omitted>);
    // CMD55, application specific CMD prefix
if (no error calling wait_for_response(APP_CMD, <...>) { // CMD55 is accepted
    send_command(SD_APP_OP_COND, <voltage range>, <...>);
        // ACMD41, to set voltage range for memory part or SD card
    wait_for_response(SD_APP_OP_COND); // voltage range is set
    if (card type is UNKNOWN) label the card as SD;
    return;
} // end of if (no error ...
else if (errors other than timeout occur) { // command/response pair is corrupted
    respond to it by program specific manner;
} // of else if (response timeout)
else { // CMD55 is refused, it must be MMC or CE-ATA card
    if (card is already labeled as SD Combo) { // change label
        re-label the card as SDIO;
        ignore the error or report it;
        return; // card is identified as SDIO card
    } // of if (card is ...)
    send_command(SEND_OP_COND, <voltage range>, <...>);
    if (RESP_TIMEOUT == wait_for_response(SEND_OP_COND)) {
        // CMD1 is not accepted, either
        label the card as UNKNOWN;
        return;
    } // of if (RESP_TIMEOUT...)
    if (check for CE-ATA signature succeeded) { // the card is CE-ATA
        store CE-ATA specific info from the signature;
        label the card as CE-ATA;
    } // of if (check for CE-ATA...)
    else label the card as MMC;
} // of else
}

```

12.6.2.4 Card Registry

Card registry on MMC and SD/SDIO/SD Combo cards are different.

For the SD card, the identification process starts at a clock rate lower than 400 KHz and the power voltage higher than 2.7 V, as defined by the card specification. At this time, the SD_CMD line output drives are

push-pull drivers instead of open-drain. After the bus is activated, the host requests the card to send their valid operation conditions. The response to ACMD41 is the operation condition register of the card. The same command should be sent to all of the new cards in the system. Incompatible cards are placed into the inactive state. The host then issues the command, ALL_SEND_CID (CMD2), to each card to get its CID. Cards that are currently unidentified (that is, in ready state), send their CID number as the response. After the CID is sent by the card, the card goes into the identification state.

The host then issues Send_Relative_Addr (CMD3), requesting the card to publish a new relative card address (RCA) that is shorter than CID. This RCA is used to address the card for future data transfer operations. Once the RCA is received, the card changes its state to the standby state. At this point, if the host wants the card to have an alternative RCA number, it may ask the card to publish a new number by sending another Send_Relative_Addr command to the card. The last published RCA is the actual RCA of the card.

The host repeats the identification process with CMD2 and CMD3 for each card in the system until the last CMD2 gets no response from any of the cards in system.

For MMC operation, the host starts the card identification process in open-drain mode with the identification clock rate lower than 400 KHz, the power voltage higher than 2.7 V. The open-drain driver stages on the SD_CMD line allow parallel card operation during card identification. After the bus is activated the host requests the cards to send their valid operation conditions (CMD1). The response to CMD1 is the wired-OR operation on the condition restrictions of all cards in the system. Incompatible cards are sent into inactive state. The host then issues the broadcast command All_Send_CID (CMD2), asking all cards for their unique CID number. All unidentified cards (the cards in ready state) simultaneously start sending their CID numbers serially, while bit-wise monitoring their outgoing bit stream. Those cards, whose outgoing CID bits do not match the corresponding bits on the command line in any one of the bit periods, stop sending their CID immediately and must wait for the next identification cycle. Since the CID is unique for each card, only one card can successfully send its full CID to the host. This card then goes into identification state. Thereafter, the host issues Set_Relative_Addr (CMD3) to assign to this card a relative card address (RCA). Once the RCA is received, the card state changes to the stand-by state, and the card does not react in further identification cycles, and its output driver switches from open-drain to push-pull. The host repeats the process, namely CMD2 and CMD3, until the host receives a time-out condition to recognize completion of the identification process.

```
card_registry()
{
do { // decide RCA for each card until response timeout
    if(card is labeled as SD Combo or SDIO) { // for SDIO card like device
        send_command(SET_RELATIVE_ADDR, 0x00, <...>);
        // ask SDIO card to publish its RCA
        retrieve RCA from response;
    } // end if (card is labeled as SD Combo...)
    else if (card is labeled as SD) { // for SD card
        send_command(ALL_SEND_CID, <...>);
        if (RESP_TIMEOUT == wait_for_response(ALL_SEND_CID)) break;
        send_command(SET_RELATIVE_ADDR, <...>);
        retrieve RCA from response;
    } // else if (card is labeled as SD ...)
    else if (card is labeled as MMC or CE-ATA) { // treat CE-ATA as MMC
        send_command(ALL_SEND_CID, <...>);
        rca = 0x1; // arbitrarily set RCA, 1 here for example, this RCA is also the
```

```

        // relative address to access the CE-ATA card
        send_command(SET_RELATIVE_ADDR, 0x1 << 16, <...>);
        // send RCA at upper 16 bits
    } // end of else if (card is labeled as MMC...)
} while (response is not timeout);
}

```

12.6.3 Card Access

These sections describe the supported access modes with external cards.

12.6.3.1 Block Write

This section describes the process of writing data to external cards in block mode.

12.6.3.1.1 Normal Write

During block write (CMD24–27), one or more blocks of data are transferred from the host to the card with a CRC appended to the end of each block by the host. If the CRC fails, the card should indicate the failure on the SD_DAT line. The transferred data is discarded and not written, and all further transmitted blocks (in multi-block write mode) are ignored.

If the host uses partial blocks whose accumulated length is not block-aligned and block misalignment is not allowed (CSD parameter WRITE_BLK_MISALIGN is not set), the card detects the block misalignment error and aborts programming before the beginning of the first misaligned block. The card sets the ADDRESS_ERROR error bit in the status register, defined in the *MMC/SD Specification*, and then waits in the receive-data state for a stop command while ignoring all further data transfers. The write operation is also aborted if the host attempts to write over a write-protected area.

For MMC and SD cards, programming the CID and CSD registers does not require a previous block length setting. The transferred data is also CRC protected. If a part of the CSD or CID register is stored in the ROM, this unchangeable section must match the corresponding section of the receive buffer. If this match fails, then the card reports an error and does not change any register contents.

Some cards may require a long and unpredictable period of time to write a block of data. After receiving a block of data and completing the CRC check, the card begins writing. If its write buffer is full and unable to accept new data from a new WRITE_BLOCK command, the card holds the SD_DAT line low. The host may poll the status of the card with a SEND_STATUS command (CMD13) or other means for SDIO cards, at any time and the card responds with its status. The card status indicates whether the card can accept new data or if the write process is still in progress. The host may deselect the card by issuing CMD7 (to select a different card) to change the card into the standby state and release the SD_DAT line without interrupting the write operation. When re-selecting the card, it reactivates the busy indication by pulling SD_DAT low if programming is still in progress and the write buffer is unavailable.

For simplicity, the software flow described below incorporates the internal DMA, and the write operation is a multi-block write with Auto CMD12 enabled. For the other method (CPU polling status) and different transfer nature, the internal DMA part of the procedure should be removed and alternative steps inserted.

1. Check the card status and wait until the card is ready for data.
2. Set the card block length.

- MMC/SD cards — use SET_BLOCKLEN (CMD16)
 - SDIO cards or the I/O portion of SD Combo cards — IO_RW_DIRECT (CMD52) to set I/O block size bit field in the CCCR register (for function 0) or FBR (for functions 1–7)
3. Set eSDHC BLKATTR[BLKSIZE] to the same as the block length set in the card in step 2.
 4. Set eSDHC BLKATTR[BLKCNT] with the number of blocks to send.
 5. Disable the buffer write ready interrupt, configure the DMA setting, and enable the eSDHC DMA when sending the command with data transfer. Set XFERTYP[AC12EN].
 6. Wait for the transfer complete interrupt.
 7. Check the status bit to see if a read CRC error or any other errors occurred between sending Auto CMD12 and receiving the response.

12.6.3.1.2 Write with Pause

The write operation can be paused during the transfer. Instead of stopping the SD_CLK at any time to pause all the operations which is also inaccessible to the host driver, the driver can set PROCTL[SABGREQ] to pause the transfer between the data blocks. Since there is no timeout condition in a write operation during the data blocks, a write operation to the cards can be paused in this way and if line SD_DAT0 is not required to de-assert to release busy state, no suspend command is needed.

Similar to the flow described in [Section 12.6.3.1.1, “Normal Write,”](#) the write with pause is shown with the same type of write operations:

1. Check the card status and wait until card is ready for data.
2. Set the card block length.
 - MMC/SD cards — use SET_BLOCKLEN (CMD16)
 - SDIO cards or the I/O portion of SD Combo cards — use IO_RW_DIRECT (CMD52) to set the I/O block size bit field in CCCR register (for function 0) or FBR (for functions 1–7)
3. Set the eSDHC BLKATTR[BLKSIZE] to the same as the block length set in the card in step 2.
4. Set eSDHC BLKATTR[BLKCNT] with the number of blocks to send.
5. Disable the buffer write ready interrupt, configure the DMA setting, and enable the eSDHC DMA when sending the command with data transfer. Set XFERTYP[AC12EN].
6. Set PROCTL[SABGREQ].
7. Wait for the transfer complete interrupt.
8. Clear PROCTL[SABGREQ].
9. Check the status bit to see if a write CRC error occurred.
10. Set PROCTL[CREQ] to continue the read operation.
11. Wait for the transfer complete interrupt.
12. Check the status bit to see if a read CRC error or any other errors occurred between sending Auto CMD12 and receiving the response.

The number of blocks left during the data transfer is accessible by reading the content of BLKATTR[BLKCNT]. Due to the data transfers and setting PROCTL[SABGREQ] are concurrent, along with the delay of register read and the register setting, the actual number of blocks left may not be the same

as the value read earlier. The driver should read the value of BLKATTR[BLKCNT] after the transfer is paused and the transfer complete interrupt is received.

It is also possible that the transfer of the last block begins when the stop-at-block-gap request is sent to the buffer. In this case, the next block gap is the actual end of the transfer, and therefore, the request is ignored. The driver should treat this as a non-pause transfer and a common write operation.

When the write operation is paused, the data transfer inside the host system does not stop and the transfer remains active until the data buffer is full. Therefore, avoid using the suspend command for the SDIO card. When such command is sent, the eSDHC assumes the system switches to another function of the SDIO card and flushes the data buffer. The eSDHC reads the resume command as a normal command with a data transfer, and it is the driver's responsibility to set all the relevant registers before the transfer is resumed. If there is only one block to send when the transfer is resumed, XFERTYP[MSBSEL, BCEN, AC12EN] are set. However, the eSDHC automatically sends CMD12 to mark the end of multi-block transfer.

12.6.3.2 Block Read

This section discusses normal read and read with pause.

12.6.3.2.1 Normal Read

For block reads, the basic unit of a data transfer is a block whose maximum size is stored in areas defined in corresponding card specifications. A CRC is appended to the end of each block, ensuring data transfer integrity. CMD17, CMD18, CMD53, and so on, can initiate a block read. After completing the transfer, the card returns to the transfer state.

For multi-block reads, data blocks are continuously transferred until a stop command is issued. If the host uses partial blocks whose accumulated length is not block aligned and blocks misalignment is not allowed, the card which does not support partial block length, should detect the block misalignment at the beginning of the first misaligned block and report the error, depending on its card type.

For simplicity, the software flow described below incorporates the internal DMA, and the read operation is a multi-block read with Auto CMD12 enabled. For the other method (CPU polling status) and different transfer nature, the internal DMA part should be removed and the alternative steps are straightforward.

1. Check the card status and wait until the card is ready for data.
2. Set the card block length.
 - MMC/SD cards — use SET_BLOCKLEN (CMD16)
 - SDIO cards or the I/O portion of SD Combo cards — use IO_RW_DIRECT (CMD52) to set IO block size bit field in CCCR register (for function 0) or FBR (for functions 1–7)
3. Set eSDHC BLKATTR[BLKSIZE] to the same as the block length set in the card in step 2.
4. Set eSDHC BLKATTR[BLKCNT] with the number of blocks to send.
5. Disable the buffer read ready interrupt, configure the DMA setting, and enable the eSDHC DMA when sending the command with data transfer. Set XFERTYP[AC12EN].
6. Wait for the transfer complete interrupt.
7. Check the status bit to see if a read CRC error or any other errors occurred between sending Auto CMD12 and receiving the response.

12.6.3.2.2 Read with Pause

In general, the read operation is not able to pause. Only the SDIO card (and SD Combo card working under I/O mode) supporting the read wait feature can pause during the read operation. If the SDIO card supports read wait (CCCR[SRW] = 1), the driver can set PROCTL[SABGREQ] to pause the transfer between the data blocks. Before setting SABGREQ, PROCTL[RWCTL] must be set. Otherwise, the eSDHC does not assert the read wait signal during the block gap and data corruption occurs. It is recommended to set the RWCTL bit once the read wait capability of the SDIO card is recognized.

Similar to the flow described in [Section 12.6.3.2.1, “Normal Read,”](#) the read with pause is shown with the same type of read operations:

1. Check CCCR[SRW] in the SDIO card to confirm the card supports read wait.
2. Set PROCTL[RWCTL].
3. Check the card status and wait until the card is ready for data.
4. Set the card block length.
 - MMC/SD cards — use SET_BLOCKLEN (CMD16)
 - SDIO cards or the I/O portion of SD Combo cards — use IO_RW_DIRECT (CMD52) to set IO block size bit field in CCCR register (for function 0) or FBR (for functions 1–7)
5. Set eSDHC BLKATTR[BLKSIZE] to the same as the block length set in the card in Step 2.
6. Set eSDHC BLKATTR[BLKCNT] with the number of blocks to send.
7. Disable the buffer read ready interrupt, configure the DMA setting, and enable the eSDHC DMA when sending the command with data transfer. Set XFERTYP[AC12EN].
8. Set PROCTL[SABGREQ].
9. Wait for the transfer complete interrupt.
10. Clear PROCTL[SABGREQ].
11. Check the status bit to see if a read CRC error occurred.
12. Set PROCTL[CREQ] to continue the read operation.
13. Wait for the transfer complete interrupt.
14. Check the status bit to see if a read CRC error or any other errors occurred between sending Auto CMD12 and receiving the response.

Similar to the write operation, it is possible to meet the ending block of the transfer when paused. In this case, the eSDHC ignores the stop-at-block-gap request and treats it as a command read operation.

Unlike the write operation, there is no remaining data inside the buffer when the transfer is paused. All data received before the pause is transferred to the host system. Whether or not a suspend command is sent, the internal data buffer is not flushed.

If the suspend command is sent and the transfer is later resumed by means of the resume command, the eSDHC takes the command as a normal one accompanied with data transfer, and it is left for the driver to set all the relevant registers before the transfer is resumed. If there is only one block to send when the transfer is resumed, XFERTYP[MSBSEL, BCEN] and IRQSTT[AC12EN] are set. However, the eSDHC automatically sends CMD12 to mark the end of a multi-block transfer.

12.6.3.3 Transfer Error

This section discusses the following errors:

- CRC
- Internal DMA
- Auto CMD12

12.6.3.3.1 CRC Error

At the end of a block transfer, a write CRC status error or read CRC error may occur. For this type of error, the last block received should be discarded because the integrity of the data block is not guaranteed. It is recommended to discard the following data blocks and re-transfer the block from the corrupted one. For a multi-block transfer, the host driver should issue CMD12 to abort the current process and start the transfer by a new data command. In this scenario, even when the XFERTYP[AC12EN, BCEN] are set, the eSDHC does not automatically send CMD12 because the last block is not transferred. On the other hand, if it is within the last block that CRC error occurs, Auto CMD12 is sent by the eSDHC. In this case, the driver should resend or re-obtain the last block with a single block transfer.

12.6.3.3.2 Internal DMA Error

During the data transfer with the internal DMA, if the DMA engine encounters an error on the CSB bus, the DMA operation is aborted and a DMA error interrupt is sent to the host system. When acknowledged by such an interrupt, the driver should calculate the start address of the data block where the error occurred. The start address can be calculated by either of the following methods:

- Read the DSADDR[DSADDR] field. Depending on initial value of this field and block size, the start address of the corrupted block can be obtained.
- Read the BLKATTR[BLKCNT] field. The start address of the corrupted block can be calculated by the number of blocks left, the total number to transfer, the start address of transfer, and the size of each block. However, if BCEN is not set, the contents of the block attribute register does not change and this method does not work.

When a DMA error occurs, it is recommended to abort the current transfer by means of CMD12 (for multi-block transfer), apply a reset for data, and restart the transfer from the corrupted block to recover the error.

12.6.3.3.3 Auto CMD12 Error

After the last block of a multi-block transfer is sent or received and XFERTYP[AC12EN] is set when the data transfer is initiated by the data command, the eSDHC automatically sends CMD12 to the card to stop the transfer. When an error occurs at this point, it is recommended that the host driver responds with one of the following actions (as appropriate to kind of error):

1. Auto CMD12 response timeout. It is not certain whether the command has been accepted by the card or not. The driver should clear the Auto CMD12 error status bits and resend CMD12 until it is accepted by the card.
2. Auto CMD12 response CRC error. Since CMD12 has been received by the card, the card aborts the transfer. The driver may ignore the error and clear the error status bit.

3. Auto CMD12 conflict error or not sent. The command was not sent. Therefore, the driver should send CMD12 manually.

12.6.3.4 Card Interrupt

The external cards can inform the host controller through the use of special signals. For SDIO cards, it can be the low level on the SD_DAT[1] line during a specific period. It is possible some other external interrupt behaviors can be defined. The eSDHC only monitors the SD_DAT[1] line and supports SDIO interrupts.

When an SDIO interrupt is captured by the eSDHC and the host system is informed by the eSDHC asserting its interrupt line, the interrupt service of host driver is requested.

As the interrupt source is controlled by the external card, the interrupt from the SDIO card must be served before the CINT bit is cleared. For the card interrupt handling flow, see [Section 12.5.6.3, “Card Interrupt Handling.”](#)

12.6.4 Switch Function

MMCs transferring data with a bus width other than one-bit wide is a new feature added to the *MMC Specification*. The high-speed timing mode for all card devices is also newly-defined in recent various card specifications. To enable these new features, a type of switch command should be issued by the host driver.

For SDIO cards, the high speed mode is enabled by writing to CCCR[EHS] after the CCCR[SHS] bit is confirmed. For SD cards, the high-speed mode is queried and enabled by CMD6 (with the mnemonic symbol as SWICH_FUNC); for MMCs, the high-speed mode is queried by CMD8 and enabled by CMD6 (with the mnemonic symbol as SWITCH).

The 4-bit and 8-bit bus width of MMC is also enabled by the SWITCH command, but with a different argument.

These new functions can also be disabled by software reset (for SDIO card, by setting RES bit in CCCR register; for other cards, by issuing CMD0), but such manner of restoring to normal mode is not recommended because a complete identification process is needed before the card is ready for data transfer.

For simplicity, the following flowcharts do not show a current capability check, which is recommended in the function switch process.

12.6.4.1 Query, Enable and Disable SDIO High Speed Mode

The following pseudo code shows enabling and disabling the high speed mode for SDIO using CMD52.

```
enable_sdio_high_speed_mode(void)
{
    send CMD52 to query bit SHS at address 0x13;
    if (SHS bit is '0')
    {
        report the SDIO card does not support high speed mode and return;
    }
    send CMD52 to set bit EHS at address 0x13 and read after write to confirm EHS bit is set;
    change clock divisor value or configure the system clock feeding into eSDHC to generate the
    card_clk of around 50MHz;
```

```

(data transactions like normal peers)
}
disable_sdio_high_speed_mode(void)
{
send CMD52 to clear bit EHS at address 0x13 and read after write to confirm EHS bit is cleared;
change clock divisor value or configure the system clock feeding into eSDHC to generate the
card_clk of the desired value below 25MHz;
(data transactions like normal peers)
}
    
```

12.6.4.2 Query, Enable and Disable SD High Speed Mode

The following pseudo code shows enabling and disabling the high speed mode for SD card using CMD6.

```

enable_sd_high_speed_mode(void)
{
    set BLKATTR[BLKCNT] to 1 (block), set BLKATTR[BLKSIZE] to 64 (bytes);
    send CMD6, with argument 0xFFFFF1 and read 64 bytes of data accompanying the R1
        response;
    wait data transfer done bit is set;
    check if the bit 401 of received 512 bit is set;
    if (bit 401 is '0') report the SD card does not support high speed mode and return;
    send CMD6, with argument 0x80FFFFF1 and read 64 bytes of data accompanying the R1
        response;
    check if the bit field 379~376 is 0xF;
    if (the bit field is 0xF) report the function switch failed and return;
    change clock divisor value or configure the system clock feeding into eSDHC to generate
        the card_clk of around 50MHz;
    (data transactions like normal peers)
}
disable_sd_high_speed_mode(void)
{
    set BLKCNT field to 1 (block), set BLKSIZE field to 64 (bytes);
    send CMD6, with argument 0x80FFFFF0 and read 64 bytes of data accompanying the R1
        response;
    check if the bit field 379~376 is 0xF;
    if (the bit field is 0xF) report the function switch failed and return;
    change clock divisor value or configure the system clock feeding into eSDHC to generate
        the card_clk of the desired value below 25MHz;
    (data transactions like normal peers)
}
    
```

12.6.4.3 Query, Enable and Disable MMC High Speed Mode

The following pseudo code shows enabling and disabling the high speed mode for MMC using CMD6.

```

enable_mmc_high_speed_mode(void)
{
    send CMD9 to get CSD value of MMC;
    check if the value of SPEC_VER field is 4 or above;
    if (SPEC_VER value is less than 4) report the MMC does not support high speed mode and
        return;
    set BLKCNT field to 1 (block), set BLKSIZE field to 512 (bytes);
    send CMD8 to get EXT_CSD value of MMC;
    extract the value of CARD_TYPE field to check the 'high speed mode' in this MMC is
        26MHz or 52MHz;
    send CMD6 with argument 0x1B90100;
}
    
```

```

    send CMD13 to wait card ready (busy line released);
    send CMD8 to get EXT_CSD value of MMC;
    check if HS_TIMING byte (byte number 185) is 1;
    if (HS_TIMING is not 1) report MMC switching to high speed mode failed and return;
    change clock divisor value or configure the system clock feeding into eSDHC to generate
        the card_clk of around 26MHz or 52MHz according to the CARD_TYPE;
    (data transactions like normal peers)
}

disable_mmc_high_speed_mode(void)
{
    send CMD6 with argument 0x2B90100;
    set BLKCNT field to 1 (block), set BLKSIZE field to 512 (bytes);
    send CMD8 to get EXT_CSD value of MMC;
    check if HS_TIMING byte (byte number 185) is 0;
    if (HS_TIMING is not 0) report the function switch failed and return;
    change clock divisor value or configure the system clock feeding into eSDHC to generate
        the card_clk of the desired value below 20MHz;
    (data transactions like normal peers)
}

```

12.6.4.4 Set MMC Bus Width

The following pseudo code shows how to set the bus width for MMC using CMD6.

```

change_mmc_bus_width(void)
{
    send CMD9 to get CSD value of MMC;
    check if the value of SPEC_VER field is 4 or above;
    if (SPEC_VER value is less than 4) report the MMC does not support multiple bit width
        and return;
    send CMD6 with argument 0x3B70x00; (8-bit, x=2; 4-bit, x=1; 1-bit, x=0)
    send CMD13 to wait card ready (busy line released);
    (data transactions like normal peers)
}

```

12.6.5 Commands for MMC/SD/SDIO

See [Table 12-26](#) for the list of commands for the MMC/SD/SDIO cards. Refer to the corresponding specifications for details about the command information.

Four kinds of commands control the MMC, as follows:

- Broadcast commands (bc)—no response
- Broadcast commands with response (bcr)—response from all cards simultaneously
- Addressed (point-to-point) commands (ac)—no data transfer on SD_DAT
- Addressed (point-to-point) data transfer commands (ADTC)

Table 12-26. Commands for MMC/SD/SDIO

CMD INDEX	Type	Argument	Resp	Abbreviation	Description ¹
CMD0	bc	[31–0] stuff bits	—	GO_IDLE_STATE	Resets all MMC and SD memory cards to idle state.
CMD1	bcr	[31–0] OCR without busy	R3	SEND_OP_COND	Asks all MMCs and SD memory cards in idle state to send their operation conditions register contents in the response on the SD_CMD line.
CMD2	bcr	[31–0] stuff bits	R2	ALL_SEND_CID	Asks all cards to send their CID numbers on the SD_CMD line.
CMD3 ⁽¹⁾	ac	[31–0] RCA	R1 R6(SDIO)	SET/SEND_RELATIVE_ADDR	Assigns relative address to the card.
CMD4	bc	[31–16] DSR [15–0] stuff bits	—	SET_DSR	Programs the DSR of all cards.
CMD5	bc	[31–0] OCR without busy	R4	IO_SEND_OP_COND	Asks all SDIO cards in idle state to send their operation conditions register contents in the response on the SD_CMD line.
CMD6 ⁽²⁾	adtc	[31] Mode 0– Check function 1– Switch function [30–8] Reserved for function groups 6 ~ 3 (All 0 or 0xFFFF) [7–4] Function group1 for command system [3–0] Function group2 for access mode	R1	SWITCH_FUNC	Checks switch ability (mode 0) and switch card function (mode 1). Refer to <i>SD Physical Specification version 1.1</i> for details.
CMD6 ⁽³⁾	ac	[31–26] Set to 0 [25–24] Access [23–16] Index [15–8] Value [7–3] Set to 0 [2–0] Cmd Set	R1b	SWITCH	Switches the mode of operation of the selected card or modifies the EXT_CSD registers. Refer to the <i>MultiMediaCard System Specification version 4.0 final draft 2</i> for details.
CMD7	ac	[31–16] RCA [15–0] stuff bits	R1b	SELECT/DESELECT_CARD	Command toggles a card between the stand-by and transfer states or between the programming and disconnect states. In both cases, the card is selected by its own relative address and gets deselected by any other address; address 0 deselected all.

Table 12-26. Commands for MMC/SD/SDIO (continued)

CMD INDEX	Type	Argument	Resp	Abbreviation	Description ¹
CMD8	bcr	[31:12] reserved bits [11:8] supply voltage(VHS) [7:0] check pattern	R7	SEND_IF_COND	Sends SD Memory Card interface condition, which includes host supply voltage information and asks the card whether card supports voltage. Reserved bits shall be set to '0'.
CMD8	adtc	[31–0] stuff bits	R1	SEND_EXT_CSD	The card sends its EXT_CSD register as a block of data, with block size of 512 bytes.
CMD9	ac	[31–16] RCA [15–0] stuff bits	R2	SEND_CSD	Addressed card sends its card-specific data (CSD) on the SD_CMD line.
CMD10	ac	[31–16] RCA [15–0] stuff bits	R2	SEND_CID	Addressed card sends its card-identification (CID) on the SD_CMD line.
CMD11	adtc	[31–0] data address	R1	READ_DAT_UNTIL_STOP	Reads data stream from the card starting at the given address until STOP_TRANSMISSION is received.
CMD12	ac	[31–0] stuff bits	R1b	STOP_TRANSMISSION	Forces the card to stop transmission.
CMD13	ac	[31–16] RCA [15–0] stuff bits	R1	SEND_STATUS	Addressed card sends its status register.
CMD14	Reserved				
CMD15	ac	[31–16] RCA [15–0] stuff bits	—	GO_INACTIVE_STATE	Sets the card to inactive state in order to protect the card stack against communication breakdowns.
CMD16	ac	[31–0] block length	R1	SET_BLOCKLEN	Sets the block length (in bytes) for all following block commands (read and write). Default block length is specified in the CSD.
CMD17	adtc	[31–0] data address	R1	READ_SINGLE_BLOCK	Reads a block of the size selected by the SET_BLOCKLEN command.
CMD18	adtc	[31–0] data address	R1	READ_MULTIPLE_BLOCK	Continuously transfers data blocks from card to host until interrupted by a stop command.
CMD19	Reserved				
CMD20	adtc	[31–0] data address	R1	WRITE_DAT_UNTIL_STOP	Writes data stream from the host starting at the given address until the STOP_TRANSMISSION command is received.
CMD21–23	Reserved				
CMD24	adtc	[31–0] data address	R1	WRITE_BLOCK	Writes a block of the size selected by the SET_BLOCKLEN command.

Table 12-26. Commands for MMC/SD/SDIO (continued)

CMD INDEX	Type	Argument	Resp	Abbreviation	Description ¹
CMD25	adtc	[31–0] data address	R1	WRITE_MULTIPLE_BLOCK	Continuously writes blocks of data until the STOP_TRANSMISSION command is received.
CMD26	adtc	[31–0] stuff bits	R1	PROGRAM_CID	Programming of the card identification register. This command should be issued only once per card. The card contains hardware to prevent this operation after the first programming. Normally this command is reserved for the manufacturer.
CMD27	adtc	[31–0] stuff bits	R1	PROGRAM_CSD	Programming of the programmable bits of the CSD.
CMD28	ac	[31–0] data address	R1b	SET_WRITE_PROT	If the card has write-protection features, this command sets the write protection bit of the addressed group. The properties of write protection are coded in the card-specific data (WP_GRP_SIZE).
CMD29	ac	[31–0] data address	R1b	CLR_WRITE_PROT	If the card provides write-protection features, this command clears the write protection bit of the addressed group.
CMD30	adtc	[31–0] write protect data address	R1	SEND_WRITE_PROT	If the card provides write-protection features, this command asks the card to send the status of the write-protection bits.
CMD31	Reserved				
CMD32	ac	[31–0] data address	R1	TAG_SECTOR_START	Sets the address of the first sector of the erase group.
CMD33	ac	[31–0] data address	R1	TAG_SECTOR_END	Sets the address of the last write block of the continuous range to be erased.
CMD34	ac	[31–0] data address	R1	UNTAG_SECTOR	Removes one previously selected sector from the erase selection.
CMD35	ac	[31–0] data address	R1	TAG_ERASE_GROUP_START	Sets the address of the first erase group within a range to be selected for erase.
CMD36	ac	[31–0] data address	R1	TAG_ERASE_GROUP_END	Sets the address of the last erase group within a continuous range to be selected for erase.
CMD37	ac	[31–0] data address	R1	UNTAG_ERASE_GROUP	Removes one previously selected erase group from the erase selection.
CMD38	ac	[31–0] stuff bits	R1b	ERASE	Erase all previously selected sectors.

Table 12-26. Commands for MMC/SD/SDIO (continued)

CMD INDEX	Type	Argument	Resp	Abbreviation	Description ¹
CMD39	ac	[31–16] RCA [15] register write flag [14–8] register address [7–0] register data	R4	FAST_IO	Used to write and read 8-bit (register) data fields. The command address a card and a register and provides the data for writing if the write flag is set. The R4 response contains data read from the address register. This command accesses application dependent registers which are not defined in the MMC standard.
CMD40	bcr	[31–0] stuff bits	R5	GO_IRQ_STATE	Sets the system into interrupt mode.
CMD41	Reserved				
CDM42	adtc	[31–0] stuff bits	R1b	LOCK_UNLOCK	Used to set/reset the password or lock/unlock the card. The size of the data block is set by the SET_BLOCK_LEN command.
CMD43–51	Reserved				
CMD52	ac	[31–0] stuff bits	R5	IO_RW_DIRECT	Access a single register within the total 128 Kbytes of register space in any I/O function.
CMD53	ac	[31–0] stuff bits	R5	IO_RW_EXTENDED	Access a multiple I/O register with a single command, it allows the reading or writing of a large number of I/O registers.
CMD54	Reserved				
CMD55	ac	[31–16] RCA [15–0] stuff bits	R1	APP_CMD	Indicates to the card that the next command is an application specific command rather than a standard command.
CMD56	adtc	[31–1] stuff bits [0]– RD/WR	R1b	GEN_CMD	Used either to transfer a data block to the card or to get a data block from the card for general-purpose or application-specific commands. The size of the data block is set by the SET_BLOCK_LEN command.
ACMDs should be preceded with the APP_CMD command (Commands listed below are for SD cards only. Other SD commands not listed below are not supported by this module)					
ACMD6	ac	[31–2] stuff bits [1–0] bus width	R1	SET_BUS_WIDTH	Defines the data bus width (00 = 1 bit or 10 = 4 bit bus) to be used for data transfer. The allowed data bus widths are given in DCR register.
ACMD13	adtc	[31–0] stuff bits	R1	SD_STATUS	Send the SD memory card status.
ACMD22	adtc	[31–0] stuff bits	R1	SEND_NUM_WR_SECTORS	Send the number of the written (without errors) sectors. Responds with 32 bit + CRC data block.

Table 12-26. Commands for MMC/SD/SDIO (continued)

CMD INDEX	Type	Argument	Resp	Abbreviation	Description ¹
ACMD23	ac	[31–23] stuff bits [22–0] number of blocks	R1	SET_WR_BLK_ERASE_COUNT	—
ACMD41	bcr	[31–0] OCR	R3	SD_APP_OP_COND	Asks the accessed card to send its operating condition register (OCR) content in the response on the SD_CMD line.
ACMD42	ac	—	R1	SET_CLR_CARD_DETECT	—
ACMD51	adtc	[31–0] stuff bits	R1	SEND_SCR	Reads the SD Configuration Register (SCR)

¹ Registers mentioned in this table are SD card registers.

NOTE

- CMD3 differs for MMC and SD cards
For MMCs, CMD3 is referred to as SET_RELATIVE_ADDR and has a response type R1
For SD cards, CMD3 is referred to as SEND_RELATIVE_ADDR and has a response type R6, with RCA inside
- CMD6 differs completely between high-speed MMCs and high-speed SD cards. Command SWITCH_FUNC is used for high speed SD cards.
- Command SWITCH is for high-speed MMCs. The index field can contain any value from 0–255, but only values 0–191 are valid. If the index value is in the 192–255 range, the card does not perform any modification and the status bit EXT_CSD[SWITCH_ERROR] is set. The access bits are shown in [Table 12-27](#):

Table 12-27. EXT_CSD Access Modes

Bits	Access Name	Operation
00	Command set	The command set is changed according to the command set field of the argument
01	Set bits	The bits in the pointed byte are set, according to the set bits in the value field.
10	Clear bits	The bits in the pointed byte are cleared, according to the set bits in the value field.
11	Write byte	The value field is written into the pointed byte.

- CMD8 differs for MMC and SD cards. For SD cards, CMD8 is referred to as SEND_IF_COND. For MMC cards, CMD8 is referred to as SEND_EXT_CSD.

12.7 Software Restrictions

This section discusses the software restrictions.

12.7.1 Initialization Active

The driver should not set INITA bit in System Control register when any of the command line or data lines is active, so the driver should ensure both CDIHB and CIHB bits are cleared. In order to auto clear the INITA bit, the SDCLKEN bit must be '1', otherwise no clocks can go out to the card and INITA never clears.

12.7.2 Software Polling Procedure

When polling read or write, once the software begins a buffer read or write, it must access exactly the number of times as set in the watermark level register, as if a DMA burst occurred.

12.7.3 Suspend Operation

In order to suspend the data transfer, the software must inform eSDHC that the suspend command is successfully accepted. To achieve this, after the Suspend command is accepted by the SDIO card, software must send another normal command marked as suspend command (CMDTYP bits set as '01') to inform eSDHC that the transfer is suspended.

If software needs resume the suspended transfer, it should read the value in BLKCNT register to save the remained number of blocks before sending the normal command marked as suspend, otherwise on sending such 'suspend' command, eSDHC regards the current transfer as aborted and change BLKCNT register to its original value, instead of keeping the remained number of blocks.

12.7.4 Data Port Access

When the internal DMA is not enabled and a write transaction is in operation, DATPORT (described in [Section 12.4.6, "Buffer Data Port Register \(DATPORT\)"](#)) must not be read. DATPORT also must not be used to read (or write) data by the CPU if the data will be written (or read) by the eSDHC internal DMA.

12.7.5 Multi-block Read

For pre-defined multi-block read operation, that is, the number of blocks to read has been defined by previous CMD23 for MMC, or pre-defined number of blocks in CMD53 for SDIO/SDCombo, or whatever multi-block read without abort command at card side, soft reset for data is required by eSDHC to drive the internal state machine to idle mode. Then use reset mechanism. For information on reset mechanism, see Section 3-10, "Error Recovery" in *SD Host Controller Specifications, Ver 2.0 (Jan 2007)*.

Chapter 13

DMA Engine 1

The direct memory access (DMA) is a second-generation platform module capable of performing complex data transfers with minimal intervention from a host processor through 16 programmable channels. The hardware micro-architecture includes a DMA engine, which performs source and destination address calculations, and the actual data movement operations, along with a local memory containing the transfer control descriptors (TCD) for the channels.

Figure 13-1 shows the DMA block diagram.

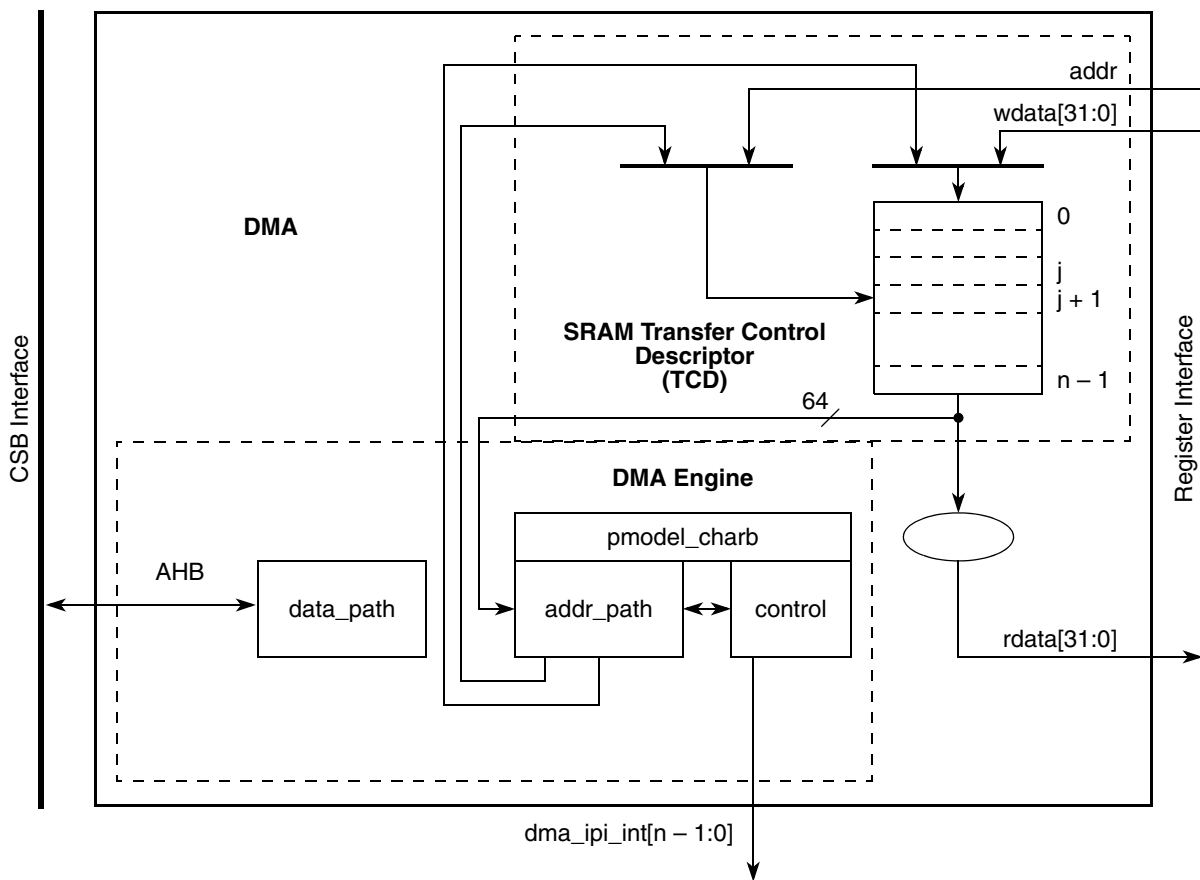


Figure 13-1. DMA Block Diagram

13.1 Overview

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

- 16 Channels
- 32-byte transfer control descriptor per channel stored in local memory
- 32 bytes of data registers, used as temporary storage to support burst transfers

Throughout this section, n is used to reference the channel number. Additionally, data sizes are defined as byte (8-bit), half-word (16-bit), word (32-bit), and double word (64-bit).

13.1.1 Features

The DMA module supports the following features:

- All data movement via dual-address transfers: read from source and write to destination
 - Programmable source, destination addresses, transfer size, plus support for enhanced addressing modes
- Transfer control descriptor organized to support two-deep, nested transfer operations
 - An *inner* data transfer loop defined by a “minor” byte transfer count
 - An *outer* data transfer loop defined by a “major” iteration count
- Channel service request via one of two methods:
 - Explicit software initiation
 - Initiation via a channel-to-channel linking mechanism for continuous transfers
 - Independent channel linking at end of minor loop and/or major loop

For both the methods, one service request per execution of the minor loop is required

- Support for fixed-priority and round-robin channel arbitration
- Channel completion reported via optional interrupt requests
 - One interrupt per channel, optionally asserted at completion of major iteration count
 - Error terminations are optionally enabled per channel, and logically summed together to form a small number of error interrupt outputs
- Support for scatter/gather DMA processing

13.2 DMAC Memory Map/Register Definition

The DMA programming model is partitioned into two sections, both mapped into the IPIslave space: the first region defines a number of registers providing control functions while the second region corresponds to the local transfer control descriptor memory. Reading an unimplemented register bit or memory location returns the value of zero. Read modified write should be used for unimplemented register bits. Any access

to a reserved memory location results in a bus error. Reserved memory locations are indicated in the memory map. [Table 13-1](#) is a 32-bit view of the DMA memory map.

Table 13-1. DMAC Register Summary

Offset	Register	Access	Reset	Section/Page
Block Base Address: 0x2_C000				
0x000	DMACR—DMA Control Register	R/W	0x0000_E400	13.2.1/13-4
0x004	DMAES—DMA Error Status Register	RO	0x0000_0000	13.3/13-5
0x008– 0x00B	Reserved	—	—	—
0x00C	DMAERQ—DMA enable request register	R/W	0x0000_0000	13.3/13-5
0x00D– 0x010	Reserved	—	—	—
0x014	DMAEEI—DMA enable error interrupt register	R/W	0x0000_0000	13.3.2/13-8
0x018	DMASERQ—DMA set enable request	R/W	0x0000_0000	13.3.3/13-9
0x019	DMACERQ—DMA clear enable request	R/W	0x0000_0000	13.3.4/13-10
0x01A	DMASEEI—DMA Set Enable Error Interrupt	R/W	0x0000_0000	13.3.3/13-9
0x01B	DMACEEI—DMA Clear Enable Error Interrupt	R/W	0x0000_0000	13.3.4/13-10
0x01C	DMACINT—DMA Clear Interrupt Request	R/W	0x0000_0000	13.3.5/13-10
0x01D	DMACERR—DMA Clear Error	R/W	0x0000_0000	13.3.6/13-11
0x01E	DMASSRT—DMA Set START Bit	R/W	0x0000_0000	13.3.7/13-12
0x01F	DMACDNE—DMA Clear DONE Status Bit	R/W	0x0000_0000	13.3.8/13-12
0x020	Reserved	—	—	—
0x024	DMAINT—DMA interrupt request register	w1c	0x0000_0000	13.3.9/13-13
0x028	Reserved	—	—	—
0x02C	DMAERR—DMA error register	w1c	0x0000_0000	13.3.10/13-14
0x030– 0x034	Reserved	—	—	—
0x038	DMAGPOR—DMA general purpose output register	R/W	0x0000_0000	13.3.11/13-15
0x03C– 0x0FC	Reserved	—	—	—
0x100– 0x13C	DCHPRI n —DMA Channel n Priority Register	R/W	0x0000_nnnn	13.3.12/13-16
0x140– 0xFFC	Reserved	—	—	—

13.2.1 DMA Control Register (DMACR)

The 32-bit DMACR defines the basic operating configuration of the DMA. There is a single group of DMA channels labeled as, “group 0,” which contain channel numbers 0–15.

Arbitration within this group can be configured to use either a fixed priority or a round robin. In fixed priority arbitration, the highest priority channel requesting service is selected to execute. The priorities are assigned by the channel priority registers (see Section 13.3.12, “DMA Channel n Priority (DCHPRIn), n = 0–15”). In round robin arbitration mode, the channel priorities are ignored and the channels within this group are cycled through without regard to priority.

Minor loop offsets are address offset values added to the final source address (saddr) or destination address (daddr) upon minor loop completion. When minor loop offsets are enabled, the minor loop offset (mloff) is added to the final source address (saddr), or the final destination address (daddr), or both prior to the addresses being written back into the TCD. If the major loop is complete, the minor loop offset is ignored and the major loop address offsets (slast and dlast_sga) are used to compute the next saddr and daddr values.

When minor loop mapping is enabled (DMACR[EMLM] = 1), TCD word2 is redefined. A portion of TCD word2 is used to specify multiple fields: a source enable bit (smloe) to specify the minor loop offset should be applied to the source address (saddr) upon minor loop completion, a destination enable bit (dmloe) to specify the minor loop offset should be applied to the destination address (daddr) upon minor loop completion, and the sign extended minor loop offset value (mloff). The same offset value (mloff) is used for both source and destination minor loop offsets. When either minor loop offset is enabled (smloe set or dmloe set), the nbytes field is reduced to 8 bits. When both minor loop offsets are disabled (smloe cleared and dmloe cleared), the nbytes field becomes a 30-bit vector.

Figure 13-2 shows the DMA control register.

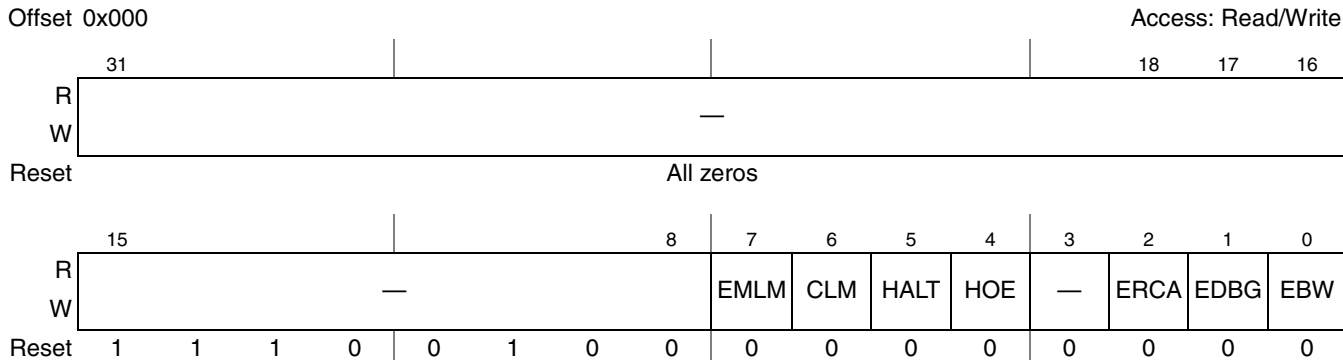


Figure 13-2. DMA Control Register (DMACR)

Table 13-2 describes the DMACR fields.

Table 13-2. DMA Control Register (DMACR) Field Descriptions

Bits	Name	Description
31–16	—	Reserved
15–8	—	Reserved

Table 13-2. DMA Control Register (DMACR) Field Descriptions (continued)

Bits	Name	Description
7	EMLM	Enable minor loop mapping. 0 Minor loop mapping disabled. TCDn.word2 is defined as a 32-bit nbytes field. 1 Minor loop mapping enabled. When set, TCDn.word2 is redefined to include individual enable fields, an offset field and the nbytes field. The individual enable fields allow the minor loop offset to be applied to the source address, the destination address, or both. The nbytes field is reduced when either offset is enabled.
6	CLM	Continuous link mode. 0 A minor loop channel link made to itself will go through channel arbitration before being activated again. 1 A minor loop channel link made to itself will not go through channel arbitration before being activated again. Upon minor loop completion, the channel will activate again if that channel has a minor loop channel link enabled and the link channel is itself. This effectively applies the minor loop offsets and restarts the next minor loop.
5	HALT	Halt DMA operations. 0 Normal operation. 1 Stall the start of any new channels. Executing channels are allowed to complete. Channel execution resumes when the HALT bit is cleared.
4	HOE	Halt on error. 0 Normal operation. 1 Any error causes the HALT bit to be set. Subsequently, all service requests will be ignored until the HALT bit is cleared.
3	—	Reserved
2	ERCA	Enable round robin channel arbitration. 0 Fixed priority arbitration is used for channel selection. 1 Round robin arbitration is used for channel selection.
1	EDBG	Enable debug. 0 The assertion of the ipg_debug input is ignored. 1 The assertion of the ipg_debug input causes the DMA to stall the start of a new channel. Executing channels are allowed to complete. Channel execution resumes when either the ipg_debug input is negated or the EDBG bit is cleared.
0	EBW	Enable buffered writes. 0 The bufferable write signal (hprot[2]) is not asserted during CSB writes. 1 The bufferable write signal (hprot[2]) is asserted on all CSB writes except for the last write sequence write sequence.

13.3 DMA Error Status (DMAES)

The DMAES register provides information concerning the last recorded channel error. Channel errors can be caused by a configuration error (an illegal setting in the transfer control descriptor or an illegal priority register setting in fixed arbitration mode) or an error termination to a bus master read or write cycle.

A configuration error is caused when the starting source or destination address, source or destination offsets, minor loop byte count and the transfer size represent an inconsistent state. The addresses and offsets must be aligned on 0-modulo-transfer_size boundaries, and the minor loop byte count must be a multiple of the source and destination transfer sizes. All source reads and destination writes must be configured to the natural boundary of the programmed transfer size respectively. In fixed arbitration mode,

a configuration error is caused by any two channel priorities. All channel priority levels within the “group 0” must be unique. If a scatter/gather operation is enabled upon channel completion, a configuration error is reported if the scatter/gather address (dlast_sga) is not aligned on a 32 byte boundary. If minor loop channel linking is enabled upon channel completion, a configuration error is reported when the link is attempted if the `TCD.citer.e_link` bit does not equal the `TCD.biter.e_link` bit. All configuration error conditions except scatter/gather and minor loop link error are reported as the channel is activated and assert an error interrupt request, if enabled. A scatter/gather configuration error is reported when the scatter/gather operation begins at major loop completion when properly enabled. A minor loop channel link configuration error is reported when the link operation is serviced at minor loop completion.

If a system bus read or write is terminated with an error, the data transfer is stopped and the appropriate bus error flag set. In this case, the state of the channel's transfer control descriptor is updated by the DMA engine with the current source address, destination address and current iteration count at the point of the fault. When a system bus error occurs, the channel is terminated after the read or write transaction which is already pipelined after errant access, has completed. If a bus error occurs on the last read prior to beginning the write sequence, the write executes using the data captured during the bus error. If a bus error occurs on the last write prior to switching to the next read sequence, the read sequence executes before the channel is terminated due to the destination bus error.

The occurrence of any type of error causes the DMA engine to immediately stop, and the appropriate channel bit in the DMA error register to be asserted. At the same time, the details of the error condition are loaded into the DMAES register. The major loop complete indicators, setting the transfer control descriptor done flag and the possible assertion of an interrupt request, are not affected when an error is detected. See [Table 13-3](#) for the DMAES definition.

Figure 13-3 shows the DMA error status register.

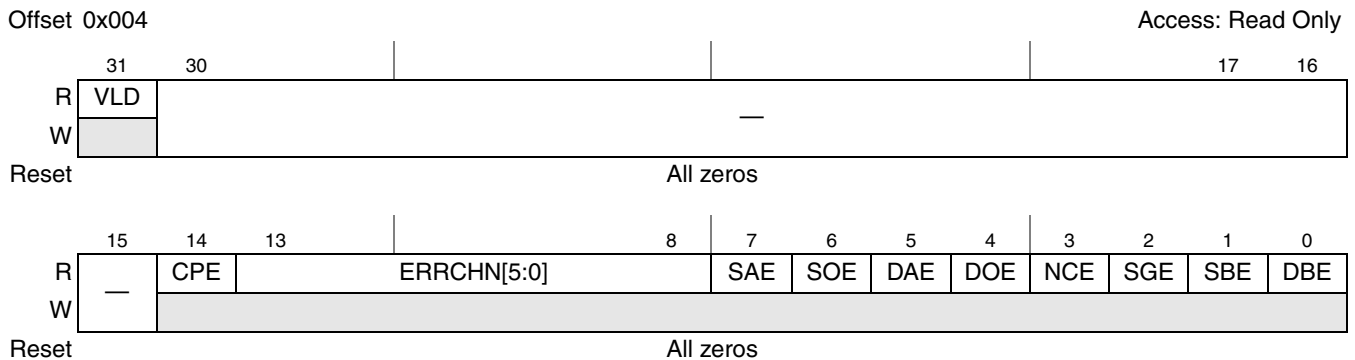


Figure 13-3. DMA Error Status Register (DMAES)

Table 13-3. DMAES Field Descriptions

Bits	Name	Value
31	VLD	Logical OR of all the DMAERR status bits. 0 No DMAERR bits are set. 1 At least one DMAERR bit is set indicating a valid error exists that has not been cleared.
30–16	—	Reserved
15	—	Reserved

Table 13-3. DMAES Field Descriptions (continued)

Bits	Name	Value
14	CPE	Channel priority error. 0 No channel priority error. 1 The last recorded error was a configuration error in the channel priorities. All channel priorities are not unique.
13–8	ERRCHN	Error channel number or cancelled channel number. The channel number of the last recorded error (excluding CPE errors) or last recorded transfer that was error cancelled.
7	SAE	Source address error. 0 No source address configuration error. 1 The last recorded error was a configuration error detected in the <code>TCD.saddr</code> field. <code>TCD.saddr</code> is inconsistent with <code>TCD.ssize</code> .
6	SOE	Source offset error. 0 No source offset configuration error. 1 The last recorded error was a configuration error detected in the <code>TCD.soff</code> field. <code>TCD.soff</code> is inconsistent with <code>TCD.ssize</code> .
5	DAE	Destination address error. 0 No destination address configuration error. 1 The last recorded error was a configuration error detected in the <code>TCD.daddr</code> field. <code>TCD.daddr</code> is inconsistent with <code>TCD.dsize</code> .
4	DOE	Destination offset error. 0 No destination offset configuration error. 1 The last recorded error was a configuration error detected in the <code>TCD.doff</code> field. <code>TCD.doff</code> is inconsistent with <code>TCD.dsize</code> .
3	NCE	Nbytes/citer configuration error. 0 No nbytes/citer configuration error. 1 The last recorded error was a configuration error detected in the <code>TCD.nbytes</code> or <code>TCD.citer</code> fields. <code>TCD.nbytes</code> is not a multiple of <code>TCD.ssize</code> and <code>TCD.dsize</code> , or <code>TCD.citer</code> is equal to zero, or <code>TCD.citer.e_link</code> is not equal to <code>TCD.biter.e_link</code> .
2	SGE	Scatter/gather configuration error. 0 No scatter/gather configuration error. 1 The last recorded error was a configuration error detected in the <code>TCD.dlast_sga</code> field. This field is checked at the beginning of a scatter/gather operation after major loop completion if <code>TCD.e_sg</code> is enabled. <code>TCD.dlast_sga</code> is not on a 32 byte boundary.
1	SBE	Source bus error. 0 No source bus error. 1 The last recorded error was a bus error on a source read.
0	DBE	Destination bus error. 0 No destination bus error. 1 The last recorded error was a bus error on a destination write.

13.3.1 DMA Enable Request Register (DMAERQ)

DMAERQ provides a bit map for the implemented channels {16,32} to enable the request signal for each channel. The state of any given channel enable is directly affected by writes to this register; it is also affected by writes to DMASERQ and DMACERQ.

DMASERQ and DMACERQ are provided so that the request enable for a single channel can easily be modified without the need to perform a read-modify-write sequence to DMAERQ. Both the DMA request input signal and this enable request flag must be asserted before a channel’s hardware service request is accepted. The state of the DMA enable request flag does not affect a channel service request made explicitly through software or a linked channel request.

As a given channel completes the processing of its major iteration count, there is a flag in the transfer control descriptor that may affect the ending state of the DMAERQ bit for that channel. If the TCD.d_req bit is set, then the corresponding DMAERQ bit is cleared, disabling the DMA request; else if the d_req bit is cleared, the state of the DMAERQ bit is unaffected.

Figure 13-4 shows the DMA enable request register.

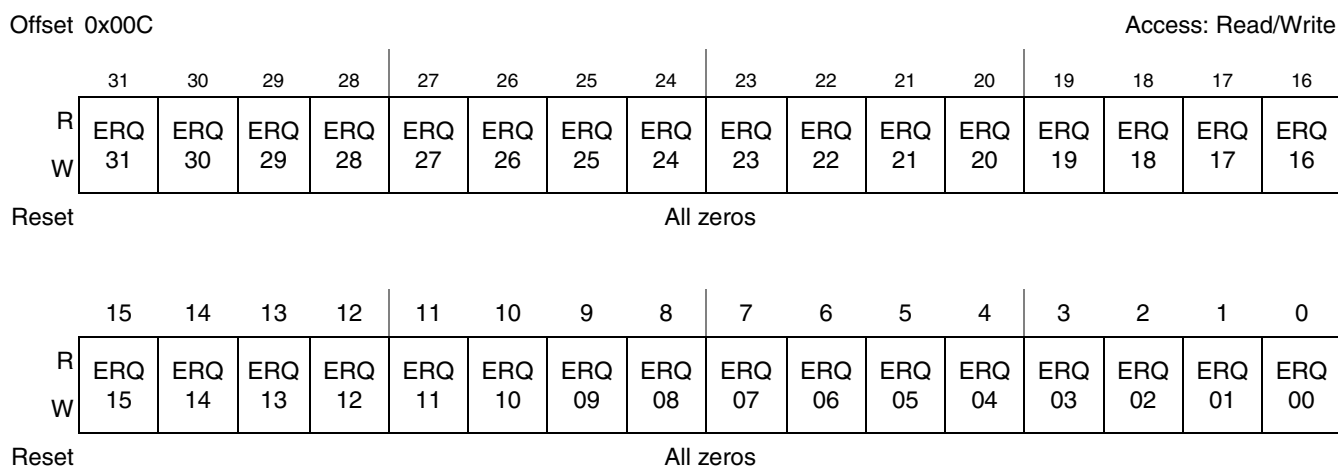


Figure 13-4. DMA Enable Request Register (DMAERQ)

See Table 13-4 for the DMAERQ definition.

Table 13-4. DMAERQ Field Descriptions

Bits	Name	Description
31-0	ERQ _n	Enable DMA request <i>n</i> . 0 The DMA request signal for channel <i>n</i> is disabled. 1 The DMA request signal for channel <i>n</i> is enabled.

13.3.2 DMA Enable Error Interrupt Register (DMAEEI)

DMAEEI provides a bit map for the 16 channels to enable the error interrupt signal for each channel. The state of any given channel’s error interrupt enable is directly affected by writes to this register; it is also affected by writes to DMASEEI and DMACEEI. DMASEEI and DMACEEI are provided so that the error interrupt enable for a single channel can easily be modified without the need to perform a read-modify-write sequence to DMAEEI.

Both the DMA error indicator and this error interrupt enable flag must be asserted before an error interrupt request for a given channel is asserted.

Figure 13-5 shows the DMA enable error interrupt register.

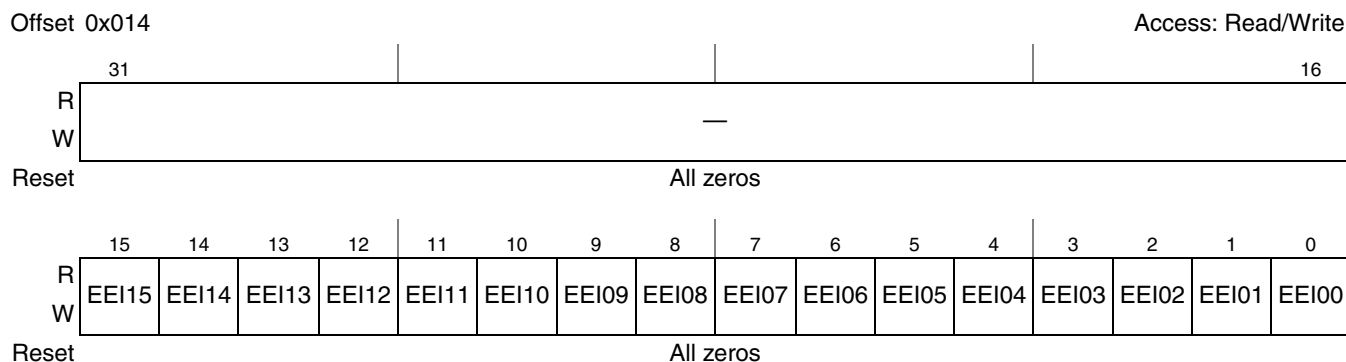


Figure 13-5. DMA Enable Error Interrupt Register (DMAEEI)

See Table 13-5 for the DMAEEI definition.

Table 13-5. DMAEEI Field Descriptions

Bits	Name	Description
31–16	—	Reserved
15–0	EEI <i>n</i>	Enable error interrupt <i>n</i> . 0 The error signal for channel <i>n</i> does not generate an error interrupt. 1 The assertion of the error signal for channel <i>n</i> generate an error interrupt request.

13.3.3 DMA Set Enable Error Interrupt (DMASEEI)

DMASEEI, shown in Figure 13-6, provides a simple memory-mapped mechanism to set a given bit in the DMAEEI register to enable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in DMAEEI to be set. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global set function, forcing the entire contents of DMAEEI to be asserted. If bit 7 is set, the command is ignored. This allows multiple-byte registers to be written as a 32-bit word. Reads of this register return all zeros.

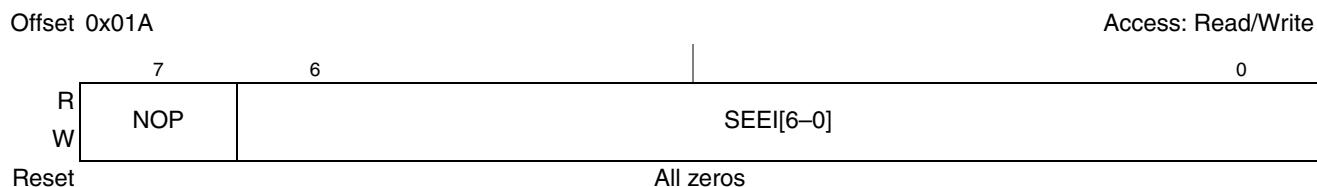


Figure 13-6. DMA Set Enable Error Interrupt Register

Table 13-6 defines the DMASEEI fields.

Table 13-6. DMASEEI Field Descriptions

Bits	Name	Description
7	NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 6–0.
6–0	SEEI	Set enable error interrupt. 0–15 Set the corresponding bit in DMAEEI. 16–63 Reserved 64–127 Set all bits in DMAEEI.

13.3.4 DMA Clear Enable Error Interrupt (DMACEEI)

The DMACEEI register, shown in Figure 13-7, provides a simple memory-mapped mechanism to clear a given bit in the DMAEEI register to disable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in DMAEEI to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing the entire contents of DMAEEI to be zeroed, disabling all DMA request inputs. If bit 7 is set, the command is ignored. This allows multiple-byte registers to be written as a 32-bit word. Reads of this register return all zeros.

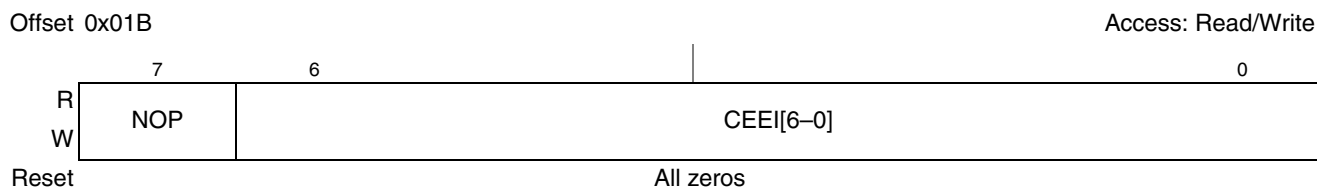


Figure 13-7. DMA Clear Enable Error Interrupt Register

Table 13-7 defines the DMACEEI fields.

Table 13-7. DMACEEI Field Descriptions

Bits	Name	Description
7	NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 6–0.
6–0	CEEI	Clear enable error interrupt. 0–15 Clear corresponding bit in DMAEEI. 16–63 Reserved 64–127 Clear all bits in DMAEEI.

13.3.5 DMA Clear Interrupt Request (DMACINT)

DMACINT, shown in Figure 13-8, provides a simple memory-mapped mechanism to clear a given bit in the DMAINT register to disable the interrupt request for a given channel. The given value on a register write causes the corresponding bit in DMAINT to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing the entire contents of

DMAINT to be zeroed, disabling all DMA interrupt requests. If bit 7 is set, the command is ignored. This allows multiple-byte registers to be written as a 32-bit word. Reads of this register return all zeros.

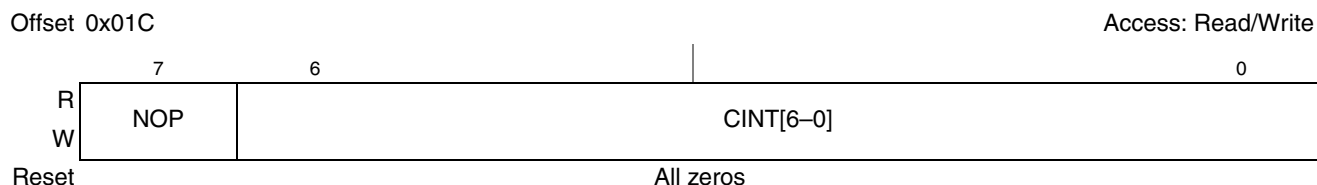


Figure 13-8. DMA Clear Interrupt Request Register

Table 13-8 defines the DMACINT fields.

Table 13-8. DMACINT Field Descriptions

Bits	Name	Description
7	NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 6-0.
6-0	CINT	Clear interrupt request. 0-15 Clear the corresponding bit in DMAINT. 16-63 Reserved 64-127 Clear all bits in DMAINT.

13.3.6 DMA Clear Error (DMACERR)

DMACEER, shown in Figure 13-9, provides a simple memory-mapped mechanism to clear a given bit in the DMAERR register to disable the error condition flag for a given channel. The given value on a register write causes the corresponding bit in DMAERR to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing the entire contents of DMAERR to be zeroed, clearing all channel error indicators. If bit 7 is set, the command is ignored. This allows multiple-byte registers to be written as a 32-bit word. Reads of this register return all zeros.



Figure 13-9. DMA Clear Error Register

Table 13-9 defines the DMACERR fields.

Table 13-9. DMACERR Field Descriptions

Bits	Name	Description
7	NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 6–0.
6–0	CERR	Clear error indicator. 0–15 Clear corresponding bit in DMAERR. 16–63 Reserved 64–127 Clear all bits in DMAERR.

13.3.7 DMA Set START Bit (DMASSRT)

DMASSRT, shown in Figure 13-10, provides a simple memory-mapped mechanism to set the START bit in the TCD of the given channel. The data value on a register write causes the START bit in the corresponding Transfer Control Descriptor to be set. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global set function, forcing all START bits to be set. If bit 7 is set, the command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeros.

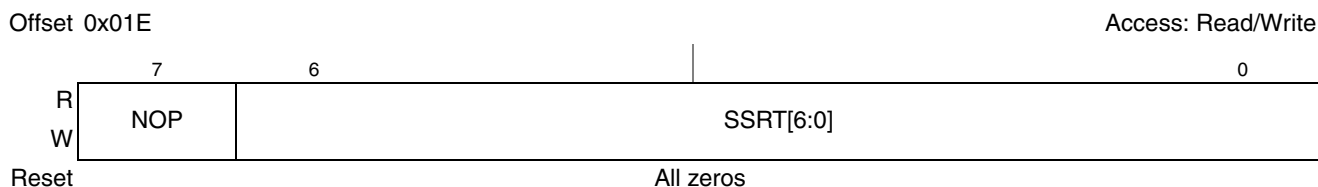


Figure 13-10. DMA Set START Bit Register

Table 13-10 defines DMASSRT fields.

Table 13-10. DMASSRT Field Descriptions

Bits	Name	Description
7	NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 6–0.
6–0	SSRT	Set START bit (channel service request). 0–15 Set the corresponding channel's TCD.start. 16–63 Reserved 64–127 Set all TCD.start bits.

13.3.8 DMA Clear DONE Status (DMACDNE)

DMACDNE, shown in Figure 13-11, provides a simple memory-mapped mechanism to clear the DONE bit in the TCD of the given channel. The data value on a register write causes the DONE bit in the corresponding transfer control descriptor to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing all DONE bits to be cleared. If bit 7 is

set, the command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeros.

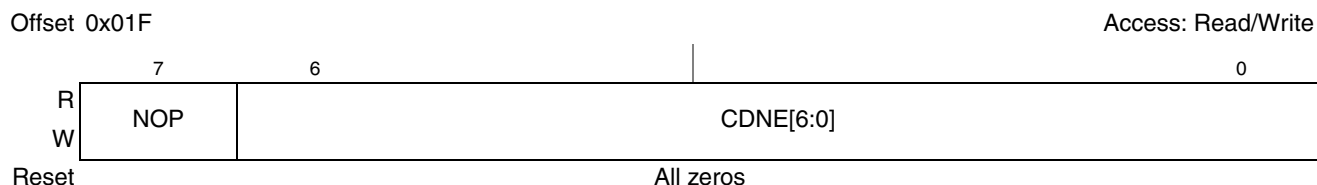


Figure 13-11. DMA Clear DONE Status Register

Table 13-11 shows the DNACDNE fields.

Table 13-11. DMACDNE Field Descriptions

Bits	Name	Description
7	NOP	No operation 0 Normal operation. 1 No operation, ignore bits 6–0.
6–0	CDNE	Clear DONE status bit. 0–15 Clear the corresponding channel's DONE bit. 16–63 Reserved 64–127 Clear all TCD DONE bits.

13.3.9 DMA Interrupt Request Register (DMAINT)

DMAINT, shown in Figure 13-12, provide a bit map for the implemented 16 channels, signaling the presence of an interrupt request for each channel. The DMA engine signals the occurrence of a programmed interrupt upon completion of a data transfer as defined in the transfer control descriptor by setting the appropriate bit in this register. The outputs of this register are directly routed to the platform’s interrupt controller. During the execution of the interrupt service routine associated with any given channel, it is software’s responsibility to clear the appropriate bit, negating the interrupt request. Typically, a write to the DMACINT register in the interrupt service routine is used for this purpose.

The state of any given channel’s interrupt request is directly affected by writes to this register; it is also affected by writes to the DMACINT register. On writes to DMAINT, a one in any bit position clears the corresponding channel’s interrupt request. A zero in any bit position has no affect on the corresponding channel’s current interrupt status. The DMACINT register is provided so the interrupt request for a single channel can easily be cleared without the need to perform a read-modify-write sequence to DMAINT.

DMA Engine 1

Offset 0x024

Access: w1c

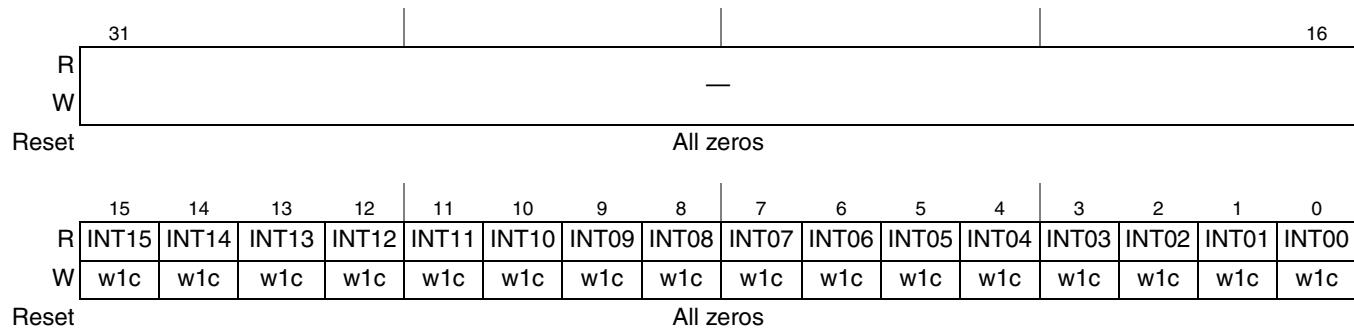


Figure 13-12. DMA Interrupt Request Register Low (DMAINT)

Table 13-12 defines the DMAINT fields.

Table 13-12. DMAINT Field Descriptions

Bits	Name	Description
31–16	—	Reserved
15–0	INT n	DMA interrupt request n (write one to clear) 0 The interrupt request for channel n is cleared. 1 The interrupt request for channel n is active.

13.3.10 DMA Error Register (DMAERR)

DMAERR, shown in Figure 13-13, provides a bit map for the 16 channels, signaling the presence of an error for each channel. The DMA engine signals the occurrence of a error condition by setting the appropriate bit in this register. The outputs of this register are enabled by the contents of the DMAEEI register, logically summed across the 16 channels to form the “group 0” error interrupt request, which is then routed to the platform’s interrupt controller. During execution of the interrupt service routine associated with any DMA errors, it is software’s responsibility to clear the appropriate bit, negating the error interrupt request; typically, a write to the DMACERR register in the interrupt service routine is used for this purpose. Recall that the normal DMA channel completion indicators, setting the transfer control descriptor done flag and the possible assertion of an interrupt request, are not affected when an error is detected.

The contents of this register can also be polled, and a non-zero value indicates the presence of a channel error regardless of the state of the DMAEEI register. The state of any given channel’s error indicators is affected by writes to this register; it is also affected by writes to DMACERR. On writes to DMAERR, a one in any bit position clears the corresponding channel’s error status; a zero in any bit position has no effect. DMACERR is provided so the error indicator for a single channel can easily be cleared.

Offset 0x02C

Access: w1c

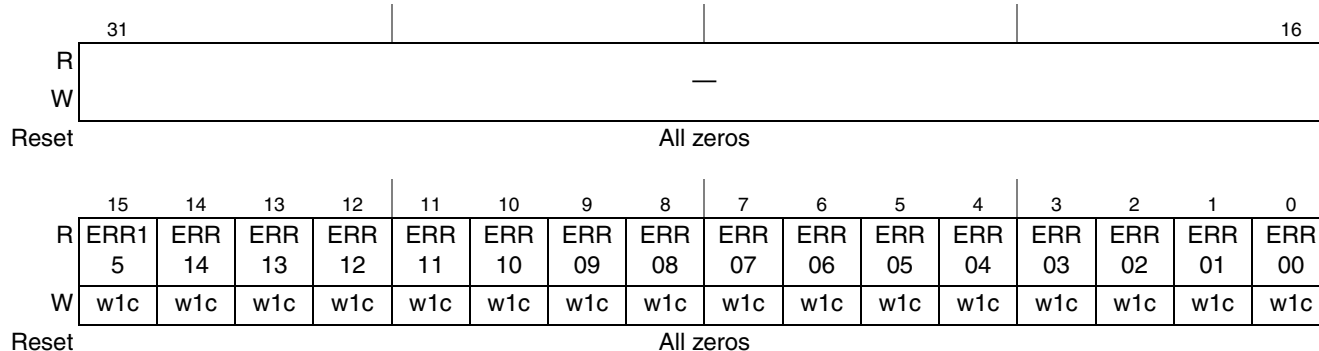


Figure 13-13. DMA Error Register (DMAERR)

See Table 13-13 for the DMAERR definition.

Table 13-13. DMAERR Field Descriptions

Bits	Name	Description
31–16	—	Reserved
15–0	INT n	DMA error n (write one to clear) 0 An error in channel n has not occurred. 1 An error in channel n has occurred.

13.3.11 DMA General Purpose Output Register (DMAGPOR)

The DMAGPOR register, as shown in Figure 13-14, provides a general purpose register in the programmer’s model that outputs the register contents.

Offset 0x038

Access: Read/Write

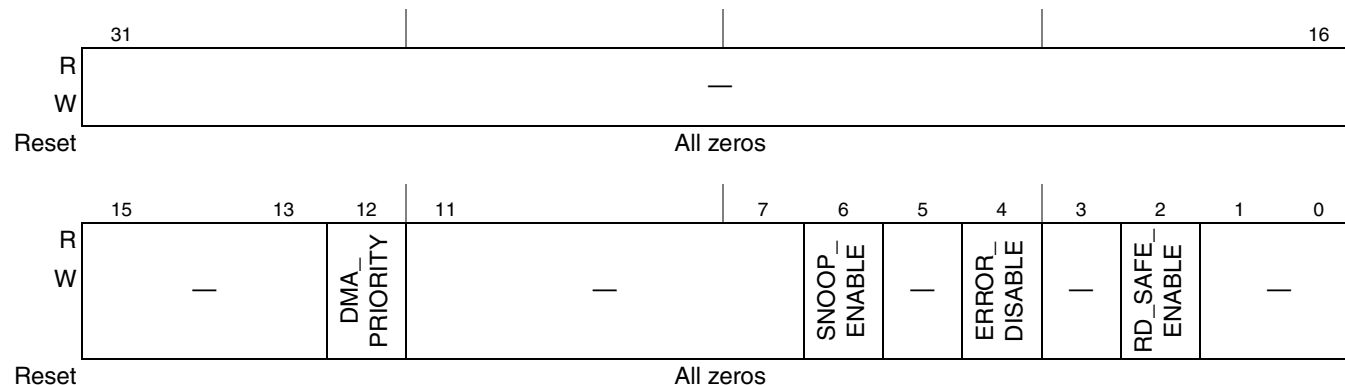


Figure 13-14. DMA General Purpose Output Register (DMAGPOR)

See [Table 13-14](#) for the DMAGPOR definition.

Table 13-14. DMAGPOR Field Descriptions

Bits	Name	Description
31–13	—	Reserved
12	DMA_PRIORITY	DMA priority. 0 Low priority 1 High priority
11–7	—	Reserved
6	SNOOP_ENABLE	Snoop attribute. 0 DMA transactions are not snooped by e300 CPU data cache 1 DMA transactions are snooped by e300 CPU data cache
5	—	Reserved
4	ERROR_DISABLE	Ignore or react to bus errors. 0 React to bus transaction errors 1 Ignore bus transaction errors
3	—	Reserved
2	RD_SAFE_ENABLE	Read Safe enable. This bit should be set only if the target of read dma operation is a well behaved memory which is not affected by the read operation and returns the same data if read again from the same location. This means that unaligned reading operation can be rounded up to enable more efficient read operations. 0 It is not safe to read more bytes that were intended 1 It is safe to read more bytes that were intended
1–0	—	Reserved

13.3.12 DMA Channel *n* Priority (DCHPRI_{*n*}), *n* = 0–15

When the fixed-priority channel arbitration mode is enabled (DMACR[ERCA] = 0), the contents of these registers define the unique priorities associated with each channel within the group. The channel priorities are evaluated by numeric value, that is, 0 is the lowest priority, 1 is the next higher priority, then 2, 3, and so on. Software must program the channel priorities with unique values, otherwise a configuration error is reported. The range of the priority value is limited to the values of 0 through 15. When read, the GRPPRI bits of the DCHPRI register reflect the current priority level of the channels. See [Figure 13-2](#) and [Table 13-2](#) for the DMACR definition.

Channel preemption is enabled on a per channel basis by setting the ECP bit in the DCHPRI register. Channel preemption allows the executing channel's data transfers to be temporarily suspended in favor of starting a higher priority channel. Once the preempting channel has completed all of its minor loop data transfers, the preempted channel is restored and resumes execution. After the restored channel completes one read/write sequence, it is again eligible for preemption. If any higher priority channel is requesting service, the restored channel is suspended and the higher priority channel is serviced. Nested preemption (attempting to preempt a preempting channel) is not supported. Once a preempting channel begins execution, it cannot be preempted. Preemption is only available when fixed arbitration is selected for channel arbitration modes.

A channel’s ability to preempt another channel can be disabled by setting the DPA bit in the DCHPRI register. When a channel’s preempt ability is disabled, that channel cannot suspend a lower priority channel’s data transfer; regardless of the lower priority channel’s ECP setting. This allows for a pool of low priority, large data moving channels to be defined. These low priority channels can be configured to not preempt each other, thus preventing a low priority channel from consuming the preempt slot normally available to a true, high priority channel.

Figure 13-15 shows the DMA clear DONE status register.

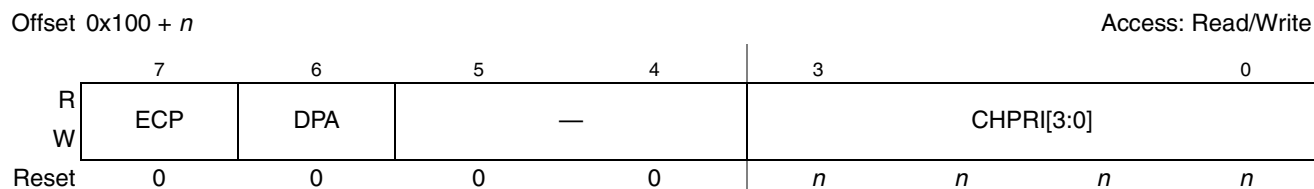


Figure 13-15. DMA Clear DONE Status Register

Table 13-15 defines the DCHPRI fields.

Table 13-15. DCHPRI n Field Descriptions

Bits	Name	Description
7	ECP	Enable channel preemption. 0 Channel n cannot be suspended by a higher priority channel's service request. 1 Channel n can be temporarily suspended by the service request of a higher priority channel.
6	DPA	Disable preempt ability. 0 Channel n can suspend a lower priority channel. 1 Channel n cannot suspend any channel, regardless of channel priority.
5-4	—	Reserved
3-0	CHPRI	Channel n arbitration priority. Channel priority when fixed-priority arbitration is enabled.

13.3.13 Transfer Control Descriptor (TCD)

Each channel requires a 32-byte transfer control descriptor for defining the desired data movement operation. The channel descriptors are stored in the local memory in sequential order: channel 0, channel

1, ..., channel [n – 1]. The definitions of the TCD are presented as eight 32-bit values. [Table 13-16](#) is a 32-bit view of the basic TCD structure.

Table 13-16. TCD 32-Bit Memory Structure

DMA Offset	TCD Field	
0x1000 + (32 x n) + 0x000	Source Address (saddr)	
0x1000 + (32 x n) + 0x004	Transfer Attributes (smod, ssize, dmod, dsize)	Signed Source Address Offset (soff)
0x1000 + (32 x n) + 0x008	Inner Minor Byte Count (nbytes)	
0x1000 + (32 x n) + 0x00C	Last Source Address Adjustment (slast)	
0x1000 + (32 x n) + 0x010	Destination Address (daddr)	
0x1000 + (32 x n) + 0x014	Current Major Iteration Count (citer)	Signed Destination Address Offset (doff)
0x1000 + (32 x n) + 0x018	Last Destination Address Adjustment/Scatter Gather Address (dlast_sga)	
0x1000 + (32 x n) + 0x01C	Beginning Major Iteration Count (biter)	Channel Control/Status (bwc, major.linkch, done, active, major.e_link, e_sg, d_req, int_half, int_maj, start)

[Figure 13-16](#) shows the TCD word 0 field.

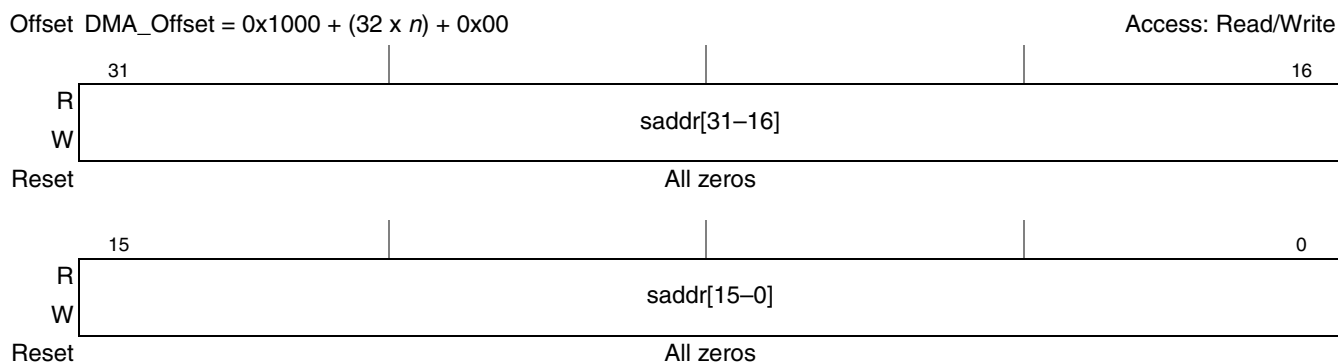


Figure 13-16. TCD Word 0 (TCDn.saddr) Field

[Table 13-17](#) describes the TCD word 0 fields.

Table 13-17. TCD Word 0 (TCDn.saddr) Field Description

Bits	Name	Description
31–0	saddr	Source address. Memory address pointing to the source data.

Figure 13-17 shows the TCD word 1 field.



Figure 13-17. TCD Word 1 (TCDn.{soff, smod, ssize, dmod, dsize}) Fields

Table 13-18 describes the TCD word 1 fields.

Table 13-18. TCD Word 1 (TCDn.{smod, ssize, dmod, dsize, soff}) Field Descriptions

Bits	Name	Description
31-27	smod	Source address modulo. 0 Source address modulo feature is disabled. Other The value defines a specific address bit which is selected to be either the value after saddr + soff calculation is performed or the original register value. This feature provides the ability to easily implement a circular data queue. For data queues requiring power-of-2 'size' bytes, the queue should be based at a 0-modulo-size address and the smod field set to the appropriate value to freeze the upper address bits. The bit select is defined as $((1 \ll \text{smod}[4:0]) - 1)$ where a resulting 1 in a bit location selects the next state address for the corresponding address bit location and a 0 selects the original register value for the corresponding address bit location. For this application, the soff is typically set to the transfer size to implement post-increment addressing with the smod function constraining the addresses to a 0-modulo-size range.
26-24	ssize	Source data transfer size. 000 8-bit 001 16-bit 010 32-bit 011 Reserved 100 Reserved 101 32-byte 110 Reserved 111 Reserved
23-19	dmod	Destination address modulo. See the smod definition.
18-16	dsize	Destination data transfer size. See the ssize definition.
15-0	soff	Source address signed offset. Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.

When minor loop mapping (DMACR[EMLM] = 1) is enabled, TCD word2, shown in Figure 13-18, is redefined as four fields: a source minor loop offset enable, a destination minor loop offset enable, a minor loop offset field, and an nbytes field.

DMA Engine 1

Offset DMA_Offset = 0x1000 + (32 x n) + 0x08

Access: Read/Write

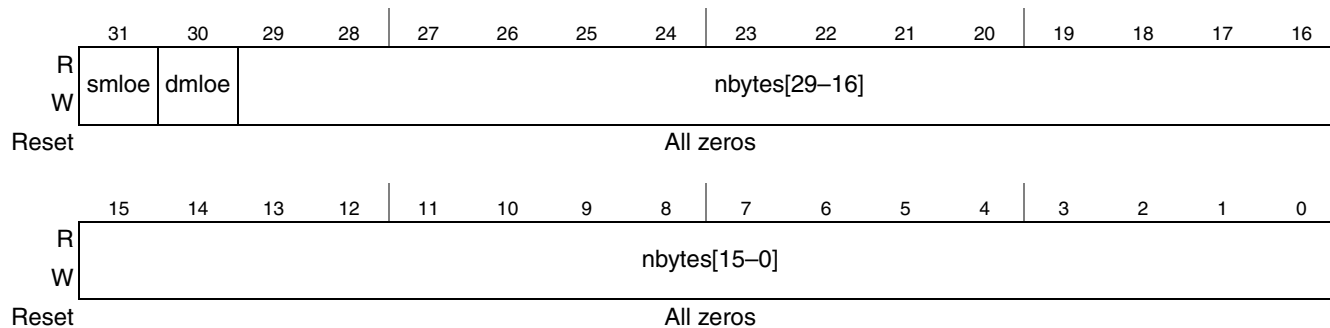


Figure 13-18. TCD Word 2 (TCDn.nbytes) Field

Table 13-19 describes the TCD word 2 fields.

Table 13-19. TCD Word 2 (TCD.nbytes) Field Description

Bits	Name	Description
31	smloe	Source minor loop offset enable. This flag selects whether the minor loop offset is applied to the source address upon minor loop completion. 0 The minor loop offset is not applied to the saddr. 1 The minor loop offset is applied to the saddr.
30	dmloe	Destination minor loop offset enable. This flag selects whether the minor loop offset is applied to the destination address upon minor loop completion. 0 The minor loop offset is not applied to the daddr. 1 The minor loop offset is applied to the daddr.
nbytes[29-0]	nbytes[29-0]	Inner minor byte transfer count. Number of bytes to be transferred in each service request of the channel. As a channel is activated, the contents of the appropriate TCD is loaded into the DMA engine, and the appropriate reads and writes performed until the complete byte transfer count has been transferred. Once the minor count is exhausted, the current values of the saddr and daddr are written back into the local memory, the major iteration count is decremented and restored to the local memory. If the major iteration count is completed, additional processing is performed. The nbytes value 0x0000_0000 is interpreted as 0x1_0000_0000, thus specifying a 4GB transfer.

Figure 13-19 shows the TCD word 3 field.

Offset DMA_Offset = 0x1000 + (32 x n) + 0x0C

Access: Read/Write

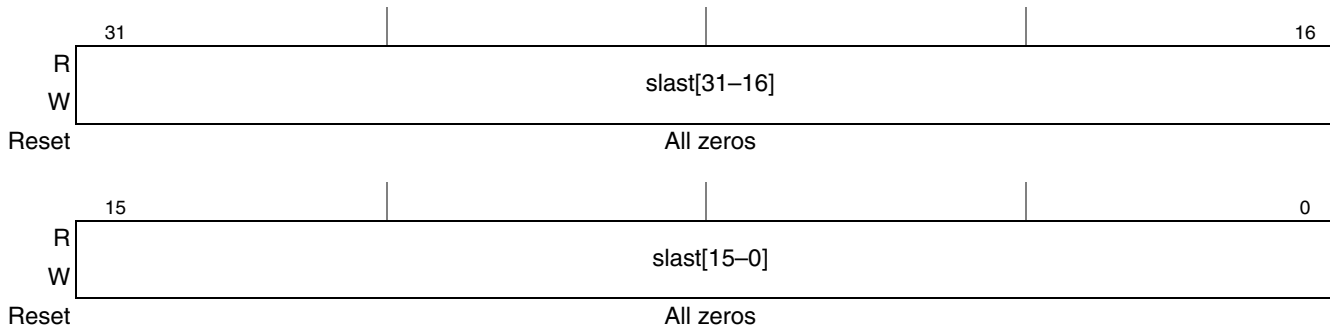


Figure 13-19. TCD Word 3 (TCDn.slast) Field

Table 13-20 describes the TCD word 3 fields.

Table 13-20. TCD Word 3 (TCD.slant) Field Descriptions

Bits	Name	Description
31-0	slant	Last source address adjustment. Adjustment value added to the source address at the completion of the outer major iteration count. This value can be applied to 'restore' the source address to the initial value, or adjust the address to reference the next data structure.

Figure 13-20 shows the TCD word 4 field.

Offset DMA_Offset = 0x1000 + (32 x n) + 0x10

Access: Read/Write

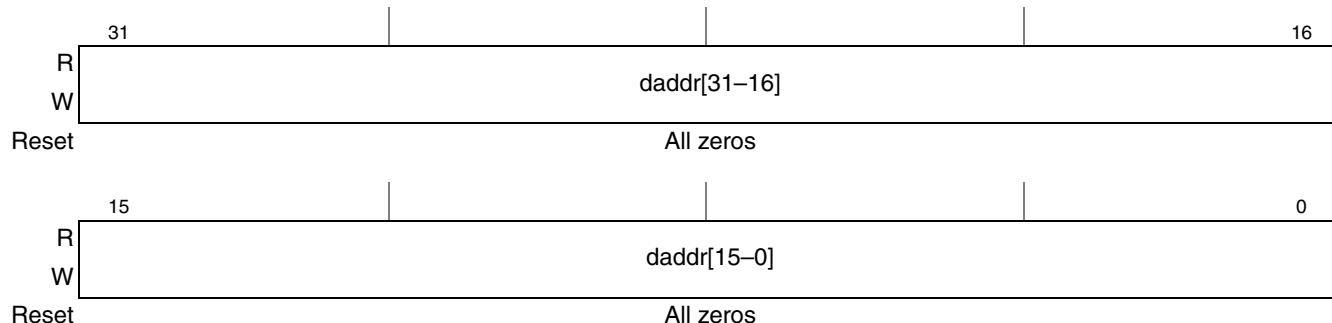


Figure 13-20. TCD Word 4 (TCDn.daddr) Field

Table 13-21 describes the TCD word 4 fields.

Table 13-21. TCD Word 4 (TCD.daddr) Field Description

Bits	Name	Description
31-0	daddr	Destination address. Memory address pointing to the destination data.

Figure 13-21 shows the TCD word 5 field.

Offset DMA_Offset = 0x1000 + (32 x n) + 0x14

Access: Read/Write

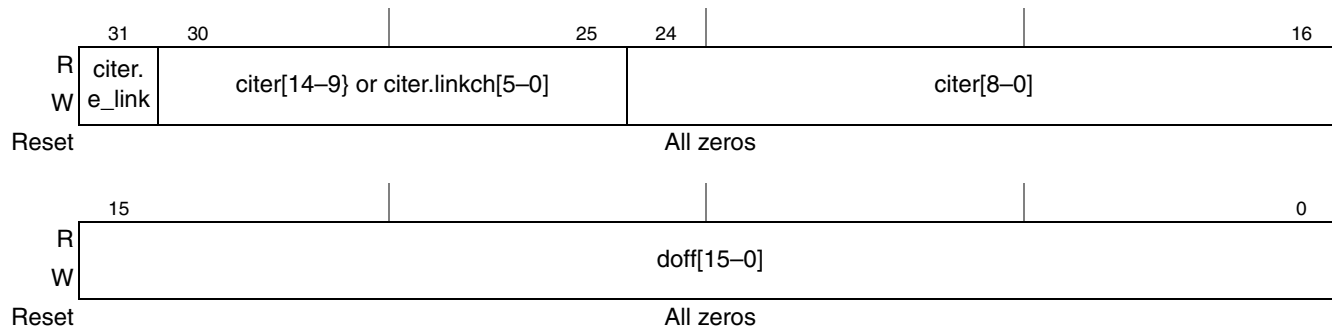


Figure 13-21. TCD Word 5 (TCDn.{citer, doff}) Fields

Table 13-22 describes the TCD word 5 fields.

Table 13-22. TCD Word 5 (TCD.{citer, doff} Field Descriptions

Bits	Name	Description
31	citer.e_link	Enable channel-to-channel linking on minor loop complete. As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by citer.linkch[5:0]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.start bit of the specified channel. If channel linking is disabled, the citer value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the major.e_link channel linking. This bit must be equal to the biter.e_link bit otherwise a configuration error is reported. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
30–25	citer[14–9] or citer.linkch[5–0]	Current major iteration count or link channel number. If (TCD.citer.e_link = 0) then no channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD word 5, bits [30:25] are used to form a 15 bit citer field. Otherwise, after the minor loop is exhausted, the DMA engine initiates a channel service request at the channel defined by citer.linkch[5:0] by setting that channel's TCD.start bit. The value contained in citer.linkch[5:0] must not exceed the number of implemented channels.
24–16	citer[8–0]	Current major iteration count. This 9 or 15-bit count represents the current major loop count for the channel. It is decremented each time the minor loop is completed and updated in the transfer control descriptor memory. Once the major iteration count is exhausted, the channel performs a number of operations (e.g., final source and destination address calculations), optionally generating an interrupt to signal channel completion before reloading the citer field from the beginning iteration count (biter) field. When the citer field is initially loaded by software, it must be set to the same value as that contained in the biter field. If the channel is configured to execute a single service request, the initial values of biter and citer should be 0x0001.
15–0	doff[15–0]	Destination address signed offset. Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.

Figure 13-22 shows the TCD word 6 field.

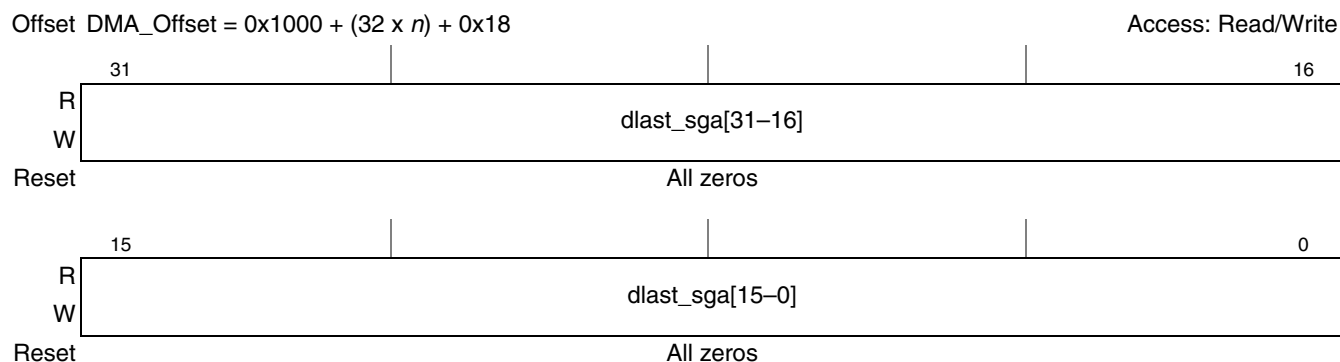


Figure 13-22. TCD Word 6 (TCDn.dlast_sga) Field

Table 13-23 describes the TCD word 6 fields.

Table 13-23. TCD Word 6 (TCD.dlast_sga) Field Descriptions

Bits	Name	Description
31-0	dlast_sga [31-0]	Last destination address adjustment or the memory address for the next transfer control descriptor to be loaded into this channel (scatter/gather). If (TCD.e_sg = 0), then Adjustment value added to the destination address at the completion of the outer major iteration count. This value can be applied to 'restore' the destination address to the initial value, or adjust the address to reference the next data structure. else, This address points to the beginning of a 0-modulo-32 region containing the next transfer control descriptor to be loaded into this channel. This channel reload is performed as the major iteration count completes. The scatter/gather address must be 0-modulo-32, else a configuration error is reported.

Figure 13-23 shows the TCD word 7 field.

$$\text{Offset DMA_Offset} = 0x1000 + (32 \times n) + 0x1C$$

Access: Read/Write

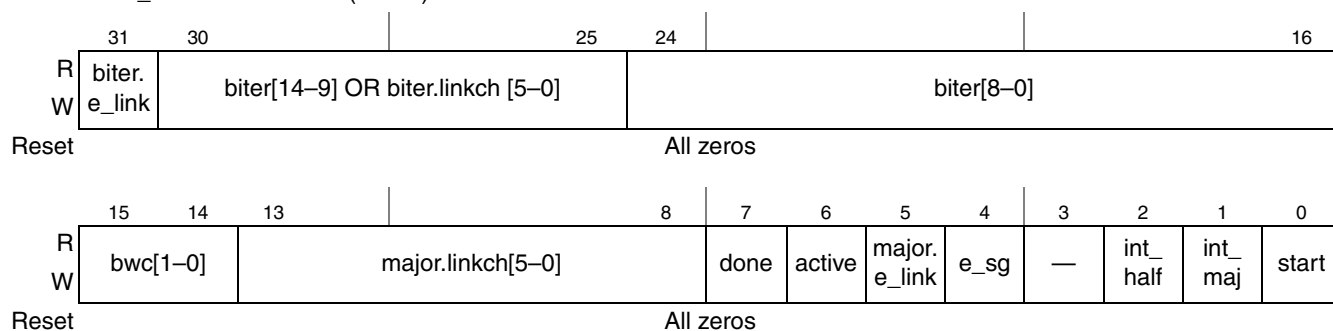


Figure 13-23. TCD Word 7 (TCDn.{biter, control/status}) Fields

Table 13-24 describes the TCD word 6 fields.

Table 13-24. TCD Word 7 (TCD.{biter, control/status}) Field Descriptions

Bits	Name	Description
31	biter.e_link	Enable channel-to-channel linking on minor loop complete. This is the initial value copied into the citer.e_link field when the major loop is completed. The citer.e_link field controls channel linking during channel execution. This bit must be equal to the citer.e_link bit otherwise a configuration error is reported. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
30-25	biter[14-9] or biter.linkch [5-0]	Beginning major iteration count or beginning link channel number. This is the initial value copied into the citer field or citer.linkch field when the major loop is completed. The citer fields control the iteration count and linking during channel execution. If (TCD.biter.e_link = 0) then No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD word 5, bits [30-25] are used to form a 15-bit biter field. Otherwise, after the minor loop is exhausted, the DMA engine initiates a channel service request at the channel defined by biter.linkch[5-0] by setting that channel's TCD.start bit. The value contained in biter.linkch[5-0] must not exceed the number of implemented channels.

Table 13-24. TCD Word 7 (TCD.{biter, control/status}) Field Descriptions (continued)

Bits	Name	Description
24–16	biter[8–0]	Beginning major iteration count. This is the initial value copied into the citer field or citer.linkch field when the major loop is completed. The citer fields controls the iteration count and linking during channel execution. This 9 or 15-bit counter presents the beginning major loop count for the channel. As the major iteration count is exhausted, the contents of the entire 16 bit biter entry is reloaded into the 16 bit citer entry. When the biter field is initially loaded by software, it must be set to the same value as that contained in the citer field. If the channel is configured to execute a single service request, the initial values of biter and citer should be 0x0001.
15–14	bwc[1–0]	Bandwidth control. This two-bit field provides a mechanism to effectively throttle the amount of bus bandwidth consumed by the DMA. In general, as the DMA processes the inner minor loop, it continuously generates read/write, read/write, ..., sequences until the minor count is exhausted. To minimize start-up latency, bandwidth control stalls are suppressed for the first two AHB bus cycles and after the last write of each minor loop. The dynamic priority elevation setting elevates the priority of the DMA as seen by the system arbiter for the executing channel. Dynamic priority elevation is suppressed during the first two AHB bus cycles. 00 No DMA engine stalls 01 dynamic priority elevation 10 DMA engine stalls for four cycles after each R/W 11 DMA engine stalls for eight cycles after each R/W
13–8	major.linkch [5–0]	Link channel number. If (TCD.major.e_link = 0) then No channel-to-channel linking (or chaining) is performed after the outer major loop counter is exhausted. Otherwise, after the major loop counter is exhausted, the DMA engine initiates a channel service request at the channel defined by major.linkch[5:0] by setting that channel's TCD.start bit. The value contained in major.linkch[5:0] must not exceed the number of implemented channels.
7	done	Channel done. This flag indicates the DMA has completed the outer major loop. It is set by the DMA engine as the citer count reaches zero; it is cleared by software, or the hardware when the channel is activated. This bit must be cleared in order to write the major.e_link or e_sg bits.
6	active	Channel active. This flag signals the channel is currently in execution. It is set when channel service begins, and is cleared by the DMA engine as the inner minor loop completes or if any error condition is detected.
5	major.e_link	Enable channel-to-channel linking on major loop complete. As the channel completes the outer major loop, this flag enables the linking to another channel, defined by major.linkch[5:0]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.start bit of the specified channel. To support the dynamic linking coherency model, this field is forced to zero when written to while the TCD.done bit is set. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
4	e_sg	Enable scatter/gather processing. As the channel completes the outer major loop, this flag enables scatter/gather processing in the current channel. If enabled, the DMA engine uses dlast_sga as a memory pointer to a 0-modulo-32 address containing a 32-byte data structure which is loaded as the transfer control descriptor into the local memory. To support the dynamic scatter/gather coherency model, this field is forced to zero when written to while the TCD.done bit is set. 0 The current channel's TCD is "normal" format. 1 The current channel's TCD specifies a scatter gather format. The dlast_sga field provides a memory pointer to the next TCD to be loaded into this channel after the outer major loop completes its execution.
3	—	Reserved

Table 13-24. TCD Word 7 (TCD.{biter, control/status}) Field Descriptions (continued)

Bits	Name	Description
2	int_half	Enable an interrupt when major counter is half complete. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the DMAINT register when the current major iteration count reaches the halfway point. Specifically, the comparison performed by the DMA engine is $(citer == (biter >> 1))$. This halfway point interrupt request is provided to support double-buffered schemes or other types of data movement where the processor needs an early indication of the transfer's progress. The halfway complete interrupt is disabled when biter values are less than two. 0 The half-point interrupt is disabled 1 The half-point interrupt is enabled.
1	int_maj	Enable an interrupt when major iteration count completes. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the DMAINT register when the current major iteration count reaches zero. 0 The end-of-major loop interrupt is disabled. 1 The end-of-major loop interrupt is enabled.
0	start	Channel start. If this flag is set, the channel is requesting service. The DMA hardware automatically clears this flag after the channel begins execution. 0 The channel is not explicitly started. 1 The channel is explicitly started via a software initiated service request.

13.4 Functional Description

This section provides an overview of the microarchitecture and functional operation of the DMA module.

13.4.1 DMA Microarchitecture

The DMA module is partitioned into two major modules: the DMA engine and the transfer control descriptor local memory. Additionally, the DMA engine is further partitioned into four submodules, which are detailed below.

- DMA engine
 - *addr_path*: This module implements registered versions of two channel transfer control descriptors: channel x and channel y, and is responsible for all the master bus address calculations. All the implemented channels provide the exact same functionality. This hardware structure allows the data transfers associated with one channel to be preempted after the completion of a read/write sequence if a higher priority channel service request is asserted while the first channel is active. Once a channel is activated, it runs until the minor loop is completed unless preempted by a higher priority channel. This capability provides a mechanism (optionally enabled by DCHPRIn[ECP]) where a large data move operation can be preempted to minimize the time another channel is blocked from execution. When any other channel is activated, the contents of its transfer control descriptor is read from the local memory and loaded into the registers of the other *addr_path.channel_{x,y}*. Once the inner minor loop completes execution, the *addr_path* hardware writes the new values for the $TCD.\{saddr, daddr, citer\}$ back into the local memory. If the major iteration count is exhausted, additional processing is performed, including the final address pointer updates, reloading the $TCD.citer$ field, and a possible fetch of the next TCD from memory as part of a scatter/gather operation.

- *data_path*: This module implements the actual bus master read/write data path. It includes 32 bytes of register storage (matching the maximum transfer size) and the necessary mux logic to support any required data alignment.
- *pmodel_charb*: This module implements the first section of DMA programming model as well as the channel arbitration logic. The programming model registers are connected to the register interface (not shown). The `dma_ipi_int[n]` outputs are also connected to this module (via the control logic).
- *control*: This module provides all the control functions for the DMA engine. For data transfers where the source and destination sizes are equal, the DMA engine performs a series of source read, destination write operations until the number of bytes specified in the inner minor loop byte count has been moved. For descriptors where the sizes are not equal, multiple access of the smaller size data are required for each reference of the larger size. As an example, if the source size references 16-bit data and the destination is 32-bit data, two reads are performed, then one 32-bit write.
- *transfer_control_descriptor* local memory
 - *memory controller*: This logic implements the required dual-ported controller, handling accesses from both the DMA engine as well as references from the register interface. As noted earlier, in the event of simultaneous accesses, the DMA engine is given priority and the register interface transaction is stalled.
 - *memory array*: The TCD is implemented using a single-ported, synchronous compiled RAM memory array.

13.4.2 DMA Basic Data Flow

The basic flow of a data transfer can be partitioned into three segments. As shown in [Figure 13-24](#), the first segment involves the channel service request. Software requests the channel service by setting the `TCD.start` bit. In the next cycle, the channel arbitration is performed, either using the fixed-priority or round-robin algorithm. After the arbitration is complete, the activated channel number is sent through the address path (`addr_path`) and converted into the required address to access the TCD local memory. Next, the TCD memory is accessed and the required descriptor read from the local memory and loaded into the DMA engine `addr_path.channel_{x,y}` registers. The TCD memory is organized 64-bits in width to

minimize the time needed to fetch the activated channel's descriptor and load it into the DMA engine `addr_path.channel_{x,y}` registers.

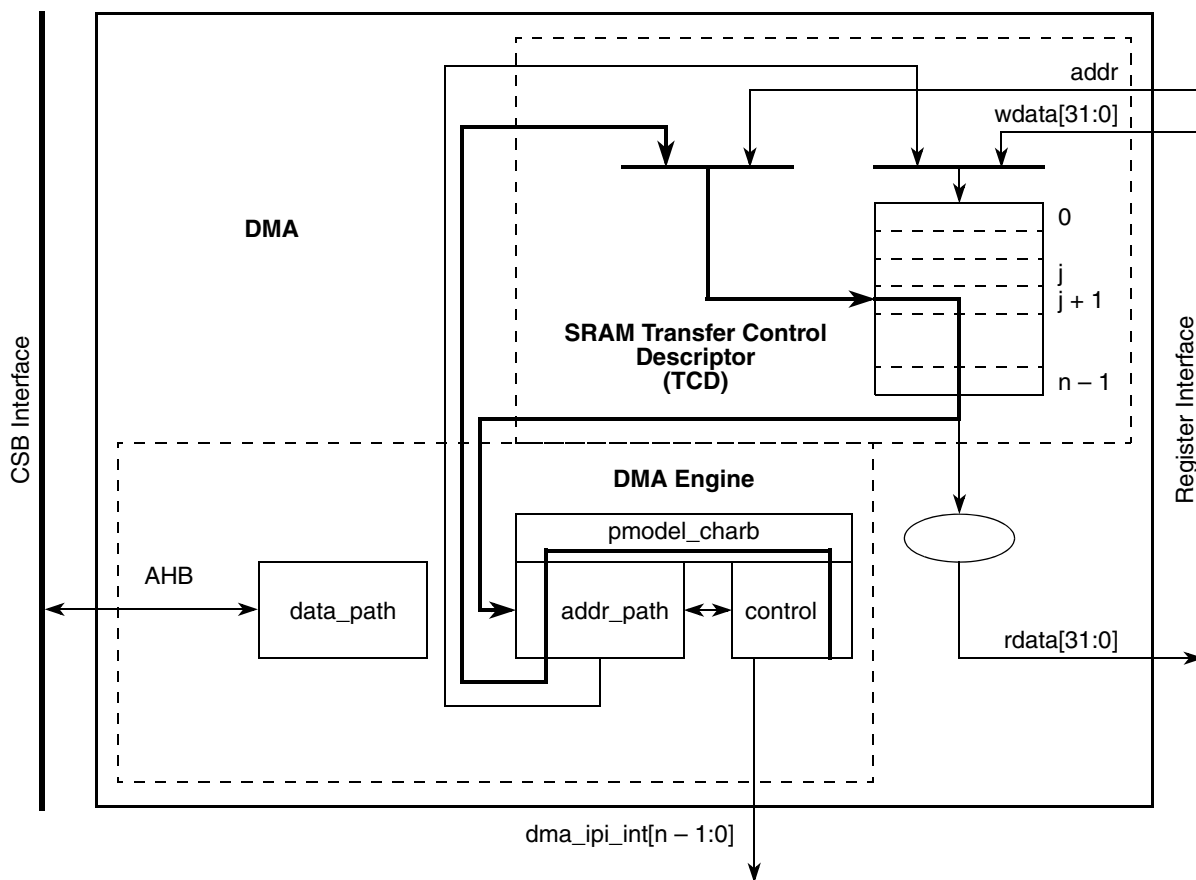


Figure 13-24. DMA Operation—Part 1

In the second part of the basic data flow as shown in [Figure 13-25](#), the modules associated with the data transfer (`addr_path`, `data_path` and `control`) sequence through the required source reads and destination writes to perform the actual data movement. The source reads are initiated and the fetched data is temporarily stored in the `data_path` module until it is gated onto the CSB bus during the destination write.

This source read/destination write processing continues until the inner minor byte count has been transferred.

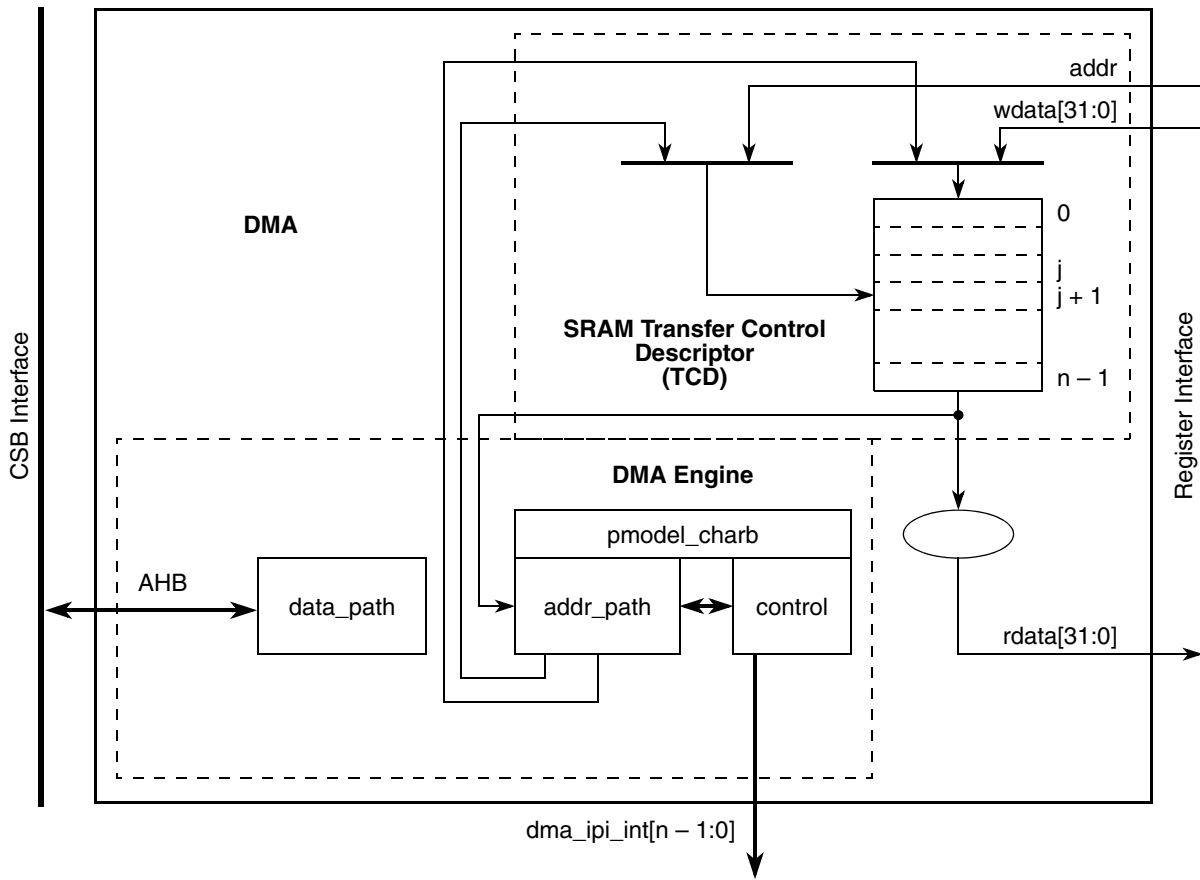


Figure 13-25. DMA Operation—Part 2

Once the inner minor byte count has been moved, the final phase of the basic data flow is performed. In this segment, the *addr_path* logic performs the required updates to certain fields in the channel's TCD, for example, *saddr*, *daddr*, *citer*. If the outer major iteration count is exhausted, then there are additional operations which are performed. These include the final address adjustments and reloading of the biter field into the *citer*. Additionally, assertion of an optional interrupt request occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the descriptor. The updates to the TCD memory and the assertion of an interrupt request are shown in

Figure 13-26.

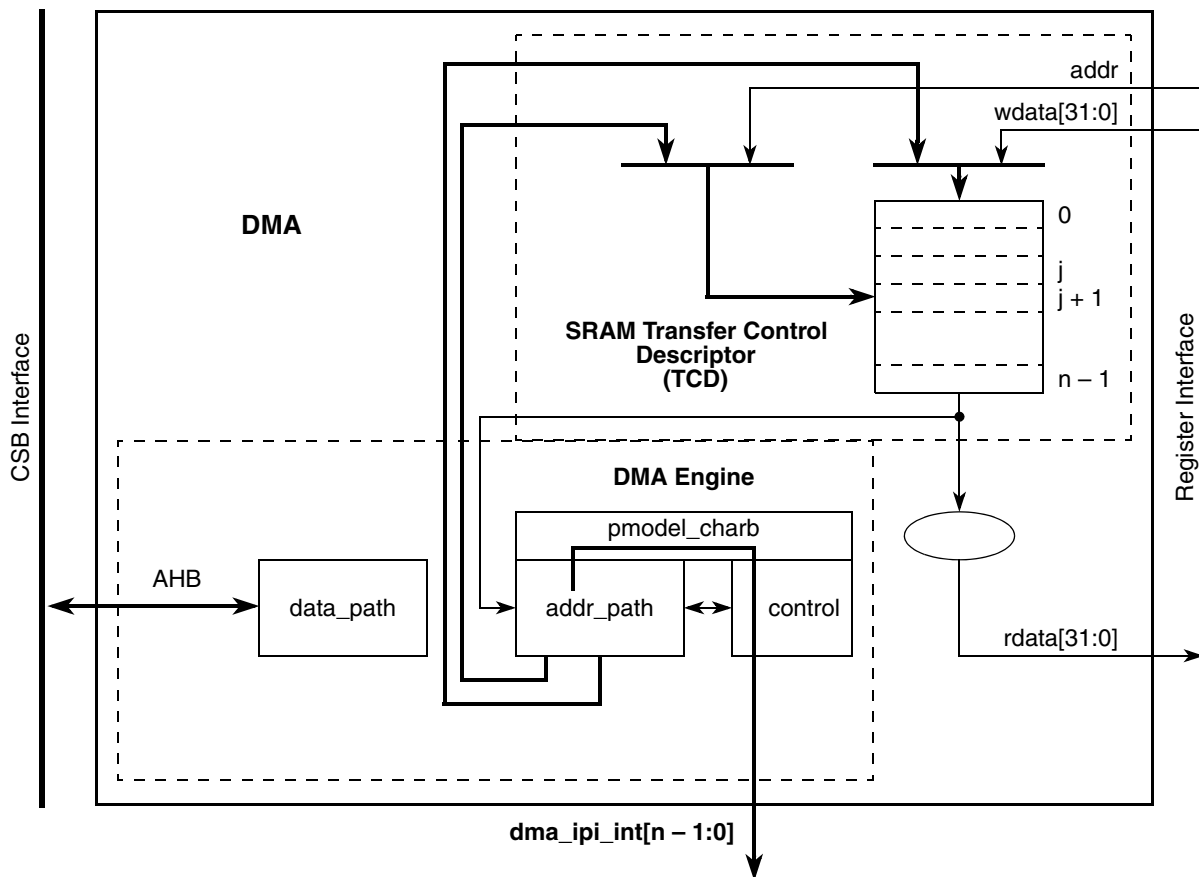


Figure 13-26. DMA Operation—Part 3

13.5 Initialization/Application Information

This section discusses the DMA initialization and programming errors.

13.5.1 DMA Initialization

The following sequence is a typical initialization of the DMA.

1. Write the DMACR register if a configuration other than the default is desired.
2. Write the channel priority levels into the DCHPRIn registers if a configuration other than the default is desired.
3. Enable error interrupts in the DMAEEI registers if so desired.
4. Write the 32 bytes TCD for each channel that may request service.
5. Request channel service by software (setting the `TCD.start` bit).

Once any channel requests service, a channel is selected for execution based on the arbitration and priority levels written into the programmer's model. The DMA engine reads the entire TCD for the selected channel into its internal address path module. As the TCD is being read, the first transfer is initiated on the

CSB bus unless a configuration error is detected. Transfers from the source (as defined by the source address, `TCD.saddr`) to the destination (as defined by the destination address, `TCD.daddr`) continue until the specified number of bytes (`TCD.nbytes`) have been transferred. When the transfer is complete, the DMA engine's local `TCD.saddr`, `TCD.daddr`, and `TCD.citer` are written back to the main TCD memory and any minor loop channel linking is performed, if enabled. If the major loop is exhausted, further post processing is executed, that is, interrupts, major loop channel linking, and scatter/gather operations, if enabled.

13.5.2 DMA Programming Errors

The DMA performs various tests on the transfer control descriptor to verify consistency in the descriptor data. Most programming errors are reported on a per channel basis with the exception of channel priority error in the DMAES register.

For all error types other than channel priority errors, the channel number causing the error is recorded in the DMAES register. If the error source is not removed before the next activation of the problem channel, the error is detected and recorded again.

In general, if priority levels are not unique, the highest channel priority that has an active request is selected, but the lowest numbered channel with that priority is selected by arbitration and executed by the DMA engine. The error interrupts and error reporting is associated with the selected channel.

13.6 DMA Transfer

This section discusses the procedures for single and multiple requests.

13.6.1 Single Request

To perform a simple transfer of n bytes of data with one activation, set the major loop to one (`TCD.citer = TCD.biter = 1`). The data transfer begins after the channel service request is acknowledged and the channel is selected to execute. Once the transfer is complete, the `TCD.done` bit is set and an interrupt is generated, if properly enabled.

For example, the following TCD entry is configured to transfer 16 bytes of data. The DMA is programmed for one iteration of the major loop transferring 16 bytes per iteration. The source memory has a byte wide memory port located at 0x1000. The destination memory has a word wide port located at 0x2000. The address offsets are programmed in increments to match the size of the transfer; one byte for the source and four bytes for the destination.

The final source and destination addresses are adjusted to return to their beginning values:

- `TCD.citer = TCD.biter = 1`
- `TCD.nbytes = 16`
- `TCD.saddr = 0x1000`
- `TCD.soff = 1`
- `TCD.ssize = 0`
- `TCD.slast = -16`
- `TCD.daddr = 0x2000`

- `TCD.doff = 4`
- `TCD.dsize = 2`
- `TCD.dlast_sga = -16`
- `TCD.int_maj = 1`
- `TCD.start = 1` (`TCD.word7` should be written last after all other fields have been initialized)
- All other TCD fields = 0

This would generate the following sequence of events:

1. Register interface write to the `TCD.start` bit requests channel service.
2. Software sets the `TCD.start` bit of the channel for activation. The channel is selected by arbitration for servicing.
3. DMA engine writes: `TCD.done = 0`, `TCD.start = 0`, `TCD.active = 1`.
4. DMA engine reads: channel TCD data from local memory to internal register file.
5. The source to destination transfers are executed as follows:
 - a) `read_byte(0x1000)`, `read_byte(0x1001)`, `read_byte(0x1002)`, `read_byte(0x1003)`
 - b) `write_word(0x2000)` -> first iteration of the minor loop
 - c) `read_byte(0x1004)`, `read_byte(0x1005)`, `read_byte(0x1006)`, `read_byte(0x1007)`
 - d) `write_word(0x2004)` -> second iteration of the minor loop
 - e) `read_byte(0x1008)`, `read_byte(0x1009)`, `read_byte(0x100a)`, `read_byte(0x100b)`
 - f) `write_word(0x2008)` -> third iteration of the minor loop
 - g) `read_byte(0x100c)`, `read_byte(0x100d)`, `read_byte(0x100e)`, `read_byte(0x100f)`
 - h) `write_word(0x200c)` -> last iteration of the minor loop -> major loop complete
6. DMA engine writes: `TCD.saddr = 0x1000`, `TCD.daddr = 0x2000`, `TCD.citer = 1` (`TCD.biter`).
7. DMA engine writes: `TCD.active = 0`, `TCD.done = 1`, `DMAINT[n] = 1`.
8. The channel retires.

The DMA goes idle or services next channel.

13.6.2 Multiple Requests

The next example is the same as previous with the exception of transferring 32 bytes by software triggered operation. The only fields that change are the major loop iteration count and the final address offsets. The DMA is programmed for two iterations of the major loop transferring 16 bytes per iteration.

- `TCD.citer = TCD.biter = 2`
- `TCD.slast = -32`
- `TCD.dlast_sga = -32`

This generates the following sequence of events:

1. Software sets the `TCD.start` bit of the channel for activation. The channel is selected by arbitration for servicing.
2. DMA engine writes: `TCD.done = 0`, `TCD.start = 0`, `TCD.active = 1`.

3. DMA engine reads: channel TCD data from local memory to internal register file.
4. The source to destination transfers are executed as follows:
 - a) read_byte(0x1000), read_byte(0x1001), read_byte(0x1002), read_byte(0x1003)
 - b) write_word(0x2000) → first iteration of the minor loop
 - c) read_byte(0x1004), read_byte(0x1005), read_byte(0x1006), read_byte(0x1007)
 - d) write_word(0x2004) → second iteration of the minor loop
 - e) read_byte(0x1008), read_byte(0x1009), read_byte(0x100a), read_byte(0x100b)
 - f) write_word(0x2008) → third iteration of the minor loop
 - g) read_byte(0x100c), read_byte(0x100d), read_byte(0x100e), read_byte(0x100f)
 - h) write_word(0x200c) → last iteration of the minor loop
5. DMA engine writes: TCD.saddr = 0x1010, TCD.daddr = 0x2010, TCD.citer = 1.
6. DMA engine writes: TCD.active = 0.
7. The channel retires → one iteration of the major loop.
The DMA goes idle or services next channel.
8. Software sets the TCD.start bit of the channel for activation. The channel is selected by arbitration for servicing.
9. DMA engine writes: TCD.done = 0, TCD.start = 0, TCD.active = 1.
10. DMA engine reads: channel TCD data from local memory to internal register file.
11. The source to destination transfers are executed as follows:
 - a) read_byte(0x1010), read_byte(0x1011), read_byte(0x1012), read_byte(0x1013)
 - b) write_word(0x2010) → first iteration of the minor loop
 - c) read_byte(0x1014), read_byte(0x1015), read_byte(0x1016), read_byte(0x1017)
 - d) write_word(0x2014) → second iteration of the minor loop
 - e) read_byte(0x1018), read_byte(0x1019), read_byte(0x101a), read_byte(0x101b)
 - f) write_word(0x2018) → third iteration of the minor loop
 - g) read_byte(0x101c), read_byte(0x101d), read_byte(0x101e), read_byte(0x101f)
 - h) write_word(0x201c) → last iteration of the minor loop → major loop complete
12. DMA engine writes: TCD.saddr = 0x1000, TCD.daddr = 0x2000, TCD.citer = 2 (TCD.biter).
13. DMA engine writes: TCD.active = 0, TCD.done = 1, DMAINT[n] = 1.
14. The channel retires → major loop complete.

The DMA goes idle or services the next channel.

13.7 TCD Status

This section discusses the two methods to test for minor loop completion and explains active channel TCD reads.

13.7.1 Minor Loop Complete

There are two methods to test for minor loop completion when using software initiated service requests. The first method is to read the `TCD.citer` field and test for a change. Another method may be extracted from the sequence shown below. The second method is to test the `TCD.start` bit AND the `TCD.active` bit. The minor loop complete condition is indicated by both bits reading zero after the `TCD.start` was written to a one. Polling the `TCD.active` bit may be inconclusive because the active status may be missed if the channel execution is short in duration. The TCD status bits execute the following sequence for a software activated channel:

1. `TCD.start = 1, TCD.active = 0, TCD.done = 0` (channel service request via software)
2. `TCD.start = 0, TCD.active = 1, TCD.done = 0` (channel is executing)
3. `TCD.start = 0, TCD.active = 0, TCD.done = 0` (channel has completed the minor loop and is idle) Or
4. `TCD.start = 0, TCD.active = 0, TCD.done = 1` (channel has completed the major loop and is idle)

The major loop complete status is explicitly indicated through the `TCD.done` bit.

The `TCD.start` bit is cleared automatically when the channel begins execution.

13.7.2 Active Channel TCD Reads

While the channel is executing, if the `TCD.saddr`, `TCD.daddr`, and `TCD.nbytes` are read, the true values of these fields are returned. The true values of `TCD.saddr`, `TCD.daddr`, and `TCD.nbytes` are the values that the DMA engine is currently using in its internal register file (true values are not the values in the TCD local memory for that channel).

The addresses (`saddr` and `daddr`) and `nbytes` (decrements to zero as the transfer progresses) can give an indication of the progress of the transfer. All other values are read back from the TCD local memory.

13.7.3 Preemption status

Preemption is only available when fixed arbitration is selected for the channel arbitration modes. A preempt-able situation is one in which a preempt-enabled channel is running and a higher priority request becomes active. When the DMA engine is not operating in the fixed channel arbitration mode, the determination of the relative priority of the actively running and the outstanding requests become undefined. Channel priorities are treated as equal (or more exactly, constantly rotating) when round-robin arbitration mode is selected.

The `TCD.active` bit for the preempted channel remains asserted throughout the preemption. The preempted channel is temporarily suspended while the preempting channel executes one iteration of the major loop. Two `TCD.active` bits set at the same time in the overall TCD map indicates a higher priority channel is actively preempting a lower priority channel.

13.8 Channel Linking

Channel linking (or chaining) is a mechanism where one channel sets the `TCD.start` bit of another channel (or itself), and initiates a service request for that channel. This operation is automatically performed by the DMA engine at the conclusion of the major or minor loop when properly enabled.

The minor loop channel linking occurs at the completion of the minor loop (or one iteration of the major loop). The `TCD.citer.e_link` field determines if a minor loop link is requested. When enabled, the channel link is made after each iteration of the major loop except for the last. When the major loop is exhausted, only the major loop channel link fields are used to determine if a channel link should be made. For example, with the initial fields of:

- `TCD.citer.e_link = 1`
- `TCD.citer.linkch = 0xC`
- `TCD.citer.value = 0x4`
- `TCD.major.e_link = 1`
- `TCD.major.linkch = 0x7`

With the bits set as mentioned above, the DMA operation executes as:

1. minor loop done -> set channel 12 `TCD.start` bit
2. minor loop done -> set channel 12 `TCD.start` bit
3. minor loop done -> set channel 12 `TCD.start` bit
4. minor loop done, major loop done -> set channel 7 `TCD.start` bit

When minor loop linking is enabled (`TCD.citer.e_link = 1`), the `TCD.citer` field uses a nine bit vector to form the current iteration count. When minor loop linking is disabled (`TCD.citer.e_link = 0`), the `TCD.citer` field uses a 15 bit vector to form the current iteration count. The bits associated with the `TCD.citer.linkch` field are concatenated onto the `citer` value to increase the range of the `citer`.

NOTE

The `TCD.citer.e_link` bit and the `TCD.biter.e_link` bit must equal or a configuration error is reported. The `citer` and `biter` vector widths must be equal in order to calculate the major loop, half-way done interrupt point.

13.9 Programming during channel execution

This section provides recommended methods to change the programming model during channel execution.

13.9.1 Dynamic priority changing

The following options are recommended for dynamically changing channel priority levels:

- Switch to round-robin channel arbitration mode, change the channel priorities, and then switch back to fixed arbitration mode.
- Disable all the channels, change the channel priorities, and then enable the appropriate channels.

13.9.2 Dynamic channel linking and dynamic scatter/gather

Dynamic channel linking and dynamic scatter/gather is the process of changing the `TCD.major.e_link` or `TCD.e_sg` bits during channel execution. These bits are read from the TCD local memory at the end of channel execution. Therefore, allows the user to enable either feature during channel execution.

Because the user is allowed to change the configuration during execution, a coherency model is needed. Consider the scenario where the user attempts to execute a dynamic channel link by enabling the `TCD.major.e_link` bit at the same time the DMA engine is retiring the channel. The `TCD.major.e_link` would be set in the programmer's model, but it would be unclear whether the actual link was made before the channel retired.

The following coherency model is recommended when executing a dynamic channel link or dynamic scatter/gather request:

1. Set the `TCD.major.e_link` bit.
2. Read back the `TCD.major.e_link` bit.
3. Test the `TCD.major.e_link` request status:
 - a) If the bit is set, the dynamic link attempt is successful.
 - b) If the bit is cleared, the attempted dynamic link did not succeed, as the channel was already retiring.

This same coherency model is true for dynamic scatter/gather operations. For both dynamic requests, the TCD local memory controller forces the `TCD.major.e_link` and `TCD.e_sg` bits to zero on any writes to a channel's `TCD.word7` once that channel's `TCD.done` bit is set indicating the major loop is complete.

NOTE

The user must clear the `TCD.done` bit before writing the `TCD.major.e_link` or `TCD.e_sg` bits. The `TCD.done` bit is cleared automatically by the DMA engine once a channel begins execution.



Chapter 14

DMA Engine 2

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 14-1](#) is a block diagram of the DMA/messaging unit.

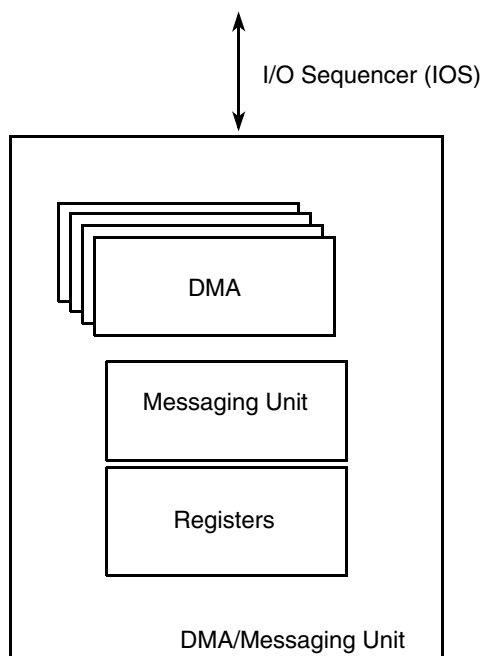


Figure 14-1. DMA/Messaging Unit Block Diagram

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.

14.1 DMA 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

14.2 DMA Memory Map/Register Definition

Table 14-1 lists the address and access of the memory map module.

Table 14-1. Module Memory Map

Offset	Register	Access	Reset	Section/Page
DMA—Block Base Address 0x0_8100				
0x0_8030	OMISR—Outbound message interrupt status register	Mixed	All zeros	14.3.1/14-3
0x0_8034	OMIMR—Outbound message interrupt mask register	R/W	All zeros	14.3.2/14-4
0x0_8050	IMR0—Inbound message register 0	R/W	All zeros	14.3.3/14-5
0x0_8054	IMR1—Inbound message register 1	R/W	All zeros	14.3.3/14-5
0x0_8058	OMR0—Outbound message register 0	R/W	All zeros	14.3.4/14-5
0x0_805C	OMR1—Outbound message register 1	R/W	All zeros	14.3.4/14-5
0x0_8060	ODR—Outbound doorbell register	R/W	All zeros	14.3.5/14-6
0x0_8068	IDR—Inbound doorbell register	R/W	All zeros	14.3.5/14-6
0x0_8080	IMISR—Inbound message interrupt status register	Mixed	All zeros	14.3.6/14-7
0x0_8084	IMIMR—Inbound message interrupt mask register	R/W	All zeros	14.3.7/14-8
0x0_8100	DMAMR0—DMA 0 mode register	R/W	All zeros	14.3.8.1/14-9
0x0_8104	DMASR0—DMA 0 status register	R/W	All zeros	14.3.8.2/14-11
0x0_8108	DMACDAR0—DMA 0 current descriptor address register	R/W	All zeros	14.3.8.3/14-12
0x0_8110	DMASAR0—DMA 0 source address register	R/W	All zeros	14.3.8.4/14-13
0x0_8118	DMADAR0—DMA 0 destination address register	R/W	All zeros	14.3.8.5/14-13
0x0_8120	DMABCR0—DMA 0 byte count register	R/W	All zeros	14.3.8.6/14-14
0x0_8124	DMANDAR0—DMA 0 next descriptor address register	R/W	All zeros	14.3.8.7/14-14
0x0_8180	DMAMR1—DMA 1 mode register	R/W	All zeros	14.3.8.1/14-9
0x0_8184	DMASR1—DMA 1 status register	R/W	All zeros	14.3.8.2/14-11
0x0_8188	DMACDAR1—DMA 1 current descriptor address register	R/W	All zeros	14.3.8.3/14-12
0x0_8190	DMASAR1—DMA 1 source address register	R/W	All zeros	14.3.8.4/14-13
0x0_8198	DMADAR1—DMA 1 destination address register	R/W	All zeros	14.3.8.5/14-13
0x0_81A0	DMABCR1—DMA 1 byte count register	R/W	All zeros	14.3.8.6/14-14
0x0_81A4	DMANDAR1—DMA 1 next descriptor address register	R/W	All zeros	14.3.8.7/14-14
0x0_8200	DMAMR2—DMA 2 mode register	R/W	All zeros	14.3.8.1/14-9
0x0_8204	DMASR2—DMA 2 status register	R/W	All zeros	14.3.8.2/14-11

Table 14-1. Module Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x0_8208	DMACDAR2—DMA 2 current descriptor address register	R/W	All zeros	14.3.8.3/14-12
0x0_8210	DMASAR2—DMA 2 source address register	R/W	All zeros	14.3.8.4/14-13
0x0_8218	DMADAR2—DMA 2 destination address register	R/W	All zeros	14.3.8.5/14-13
0x0_8220	DMABCR2—DMA 2 byte count register	R/W	All zeros	14.3.8.6/14-14
0x0_8224	DMANDAR2—DMA 2 next descriptor address register	R/W	All zeros	14.3.8.7/14-14
0x0_8280	DMAMR3—DMA 3 mode register	R/W	All zeros	14.3.8.1/14-9
0x0_8284	DMASR3—DMA 3 status register	R/W	All zeros	14.3.8.2/14-11
0x0_8288	DMACDAR3—DMA 3 current descriptor address register	R/W	All zeros	14.3.8.3/14-12
0x0_8290	DMASAR3—DMA 3 source address register	R/W	All zeros	14.3.8.4/14-13
0x0_8298	DMADAR3—DMA 3 destination address register	R/W	All zeros	14.3.8.5/14-13
0x0_82A0	DMABCR3—DMA 3 byte count register	R/W	All zeros	14.3.8.6/14-14
0x0_82A4	DMANDAR3—DMA 3 next descriptor address register	R/W	All zeros	14.3.8.7/14-14
0x0_82A8	DMAGSR—DMA general status register	R	All zeros	14.3.8.8/14-15
0x0_82B0– 0x0_82FF	Reserved	—	—	—

14.3 DMA Register Descriptions

The following sections describe the DMA/messaging unit configuration, control, and status registers.

NOTE

The registers described in this section use little-endian byte ordering. Software running on the local processor in big-endian mode must byte-swap the data. No byte swapping occurs when the registers are accessed from the PCI bus.

14.3.1 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 14-2](#) shows the OMISR fields.

Table 14-3 describes the OMIMR register.

Table 14-3. 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

14.3.3 Inbound Message Registers (IMR0–IMR1)

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 14-4 shows the IMR0 and IMR1 fields.

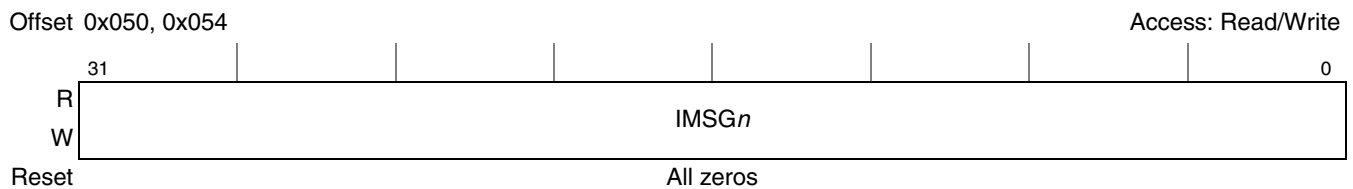


Figure 14-4. Inbound Message Registers (IMR0, IMR1)

Table 14-4 describes the IMR_n register.

Table 14-4. IMR0 and IMR1 Field Descriptions

Bits	Name	Description
31–0	IMSG _n	Inbound message <i>n</i> . Contains generic data to be passed between the local processor and external hosts.

14.3.4 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.

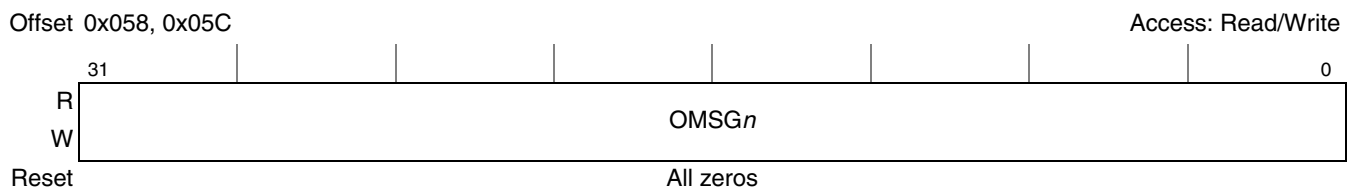


Figure 14-5. Outbound Message Registers (OMR0–OMR1)

Table 14-5 describes the OMR_n registers.

Table 14-5. OMR0 and OMR1 Field Descriptions

Bits	Name	Description
31-0	OMSG _n	Outbound message <i>n</i> . Contains generic data to be passed between the local processor and external hosts.

14.3.5 Doorbell Registers

The following sections describe the outbound and inbound doorbell registers.

14.3.5.1 Outbound Doorbell Register (ODR)

ODR is accessible from the PCI bus and the CSB in both host and agent modes. Figure 14-6 shows the ODR_n fields.

Offset: 0x060

Access: Read/Write

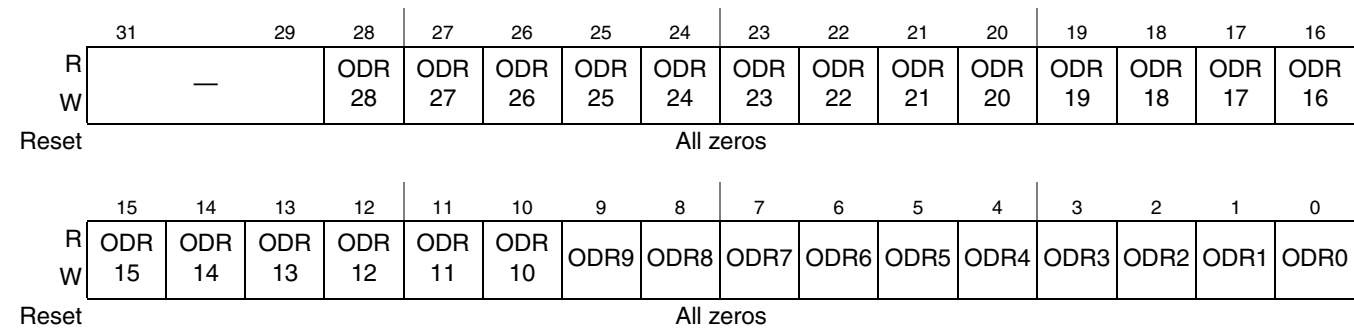


Figure 14-6. Outbound Doorbell Register (ODR)

Table 14-6 describes the ODR registers.

Table 14-6. ODR Field Descriptions

Bits	Name	Description
31-29	—	Reserved
28-0	ODR _n	Outbound doorbell <i>n</i> . 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 ($\overline{\text{PCI_INTA}}$) to be generated.)

14.3.5.2 Inbound Doorbell Register (IDR)

IDR is accessible from the PCI bus and the CSB in both host and agent modes. [Figure 14-7](#) shows the IDR fields.

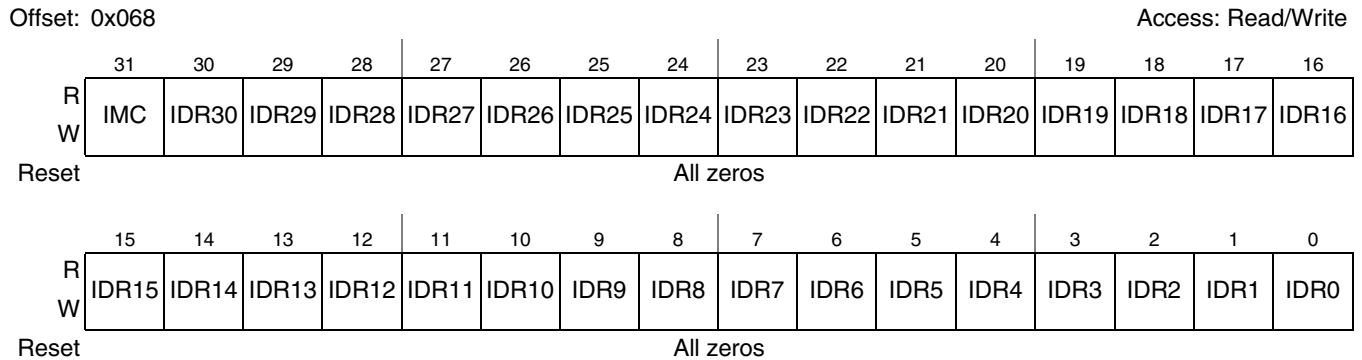


Figure 14-7. Inbound Doorbell Register (IDR)

[Table 14-7](#) describes the IDR registers.

Table 14-7. 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.

14.3.6 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.

[Figure 14-8](#) shows the IMISR fields.

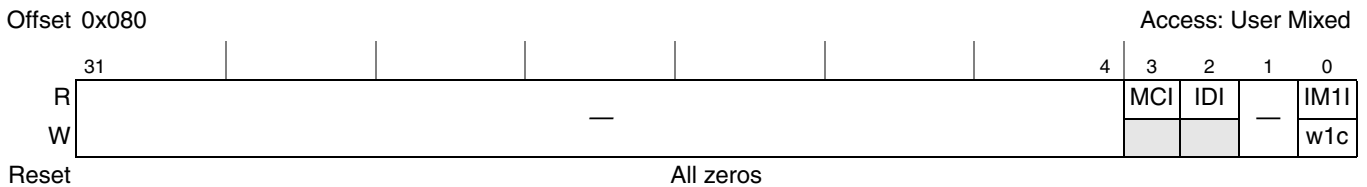


Figure 14-8. Inbound Message Interrupt Status Register (IMISR)

Table 14-8 describes the IMISR register.

Table 14-8. 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.

14.3.7 Inbound Message Interrupt Mask Register (IMIMR)

This register contains the interrupt mask of the doorbell and message register events generated by the PCI master. Figure 14-9 shows the IMIMR fields.



Figure 14-9. Inbound Message Interrupt Mask Register (IMIMR)

Table 14-9 describes the IMISR register.

Table 14-9. 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

Table 14-9. IMIMR Field Descriptions (continued)

Bits	Name	Description
1	IM1IM	Inbound message 1 interrupt mask. 0 Inbound message 1 interrupt is allowed 1 Inbound message 1 interrupt is masked. IMISR[IM1] is cleared
0	IM0IM	Inbound message 0 interrupt mask. 0 Inbound message 0 interrupt is allowed 1 Inbound message 0 interrupt is masked. IMISR[IM0] is cleared

14.3.8 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.

14.3.8.1 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 14-10](#) shows the DMAMR_n fields.

Offset: 0x100, 0x180, 0x200, 0x280

Access: Read/Write

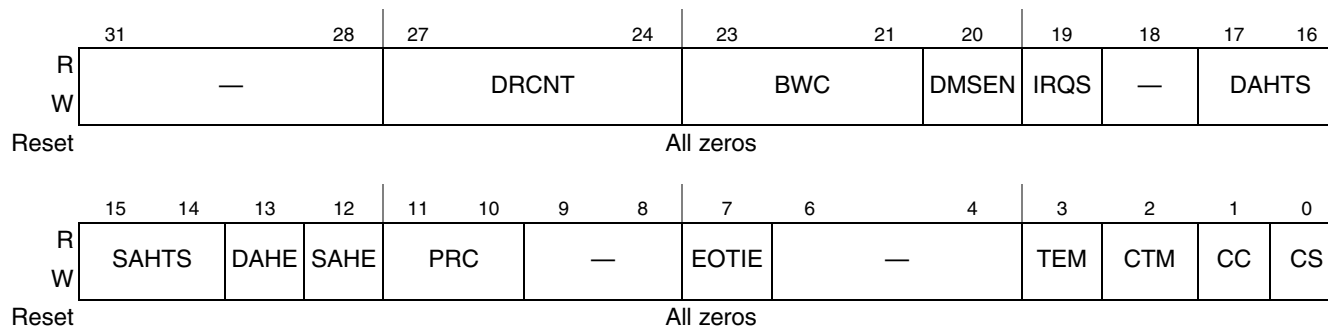


Figure 14-10. DMA Mode Register (DMAMR_n)

[Table 14-10](#) describes the DMAMR_n register.

Table 14-10. 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. 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

Table 14-10. DMAMR n Field Descriptions (continued)

Bits	Name	Description
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 PCI_INTA
18	—	Reserved
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 Reserved 01 PCI read line 10 PCI read multiple 11 Reserved

Table 14-10. DMAMR_n Field Descriptions (continued)

Bits	Name	Description
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. 1 The DMA will complete the transfer regardless of whether a transfer error occurs. Note: Regardless of the setting of TEM, if an error condition was detected during the DMA transfer, it will cause DMASR _n [TE] to be set.
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.

14.3.8.2 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 14-11](#) shows the DMASR_n fields.



Figure 14-11. DMA Status Register (DMASR_n)

Table 14-11 describes the DMASR n register.

Table 14-11. 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.
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: an error or completion of the DMA transfer. 0 No DMA transfer is currently in progress 1 A DMA transfer is currently in progress
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.

14.3.8.3 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 14-12 shows the DMACDAR n fields.

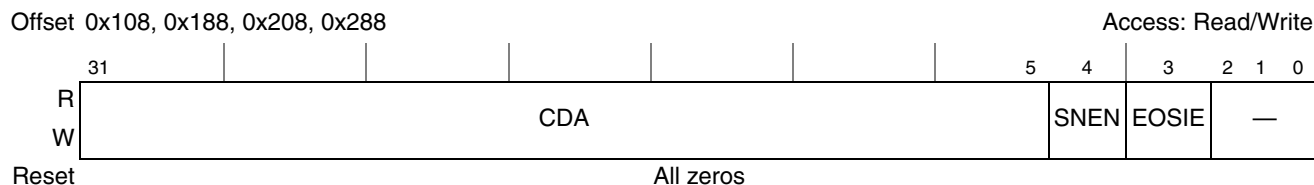


Figure 14-12. DMA Current Descriptor Address Register (DMACDAR n)

Table 14-12 describes the DMACDAR n register.

Table 14-12. 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

14.3.8.4 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 14-13](#) shows the DMASAR_n.



Figure 14-13. DMA Source Address Register (DMASAR_n)

[Table 14-13](#) describes the DMASAR_n register.

Table 14-13. 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.

14.3.8.5 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 14-14](#) shows the DMADAR_n fields.

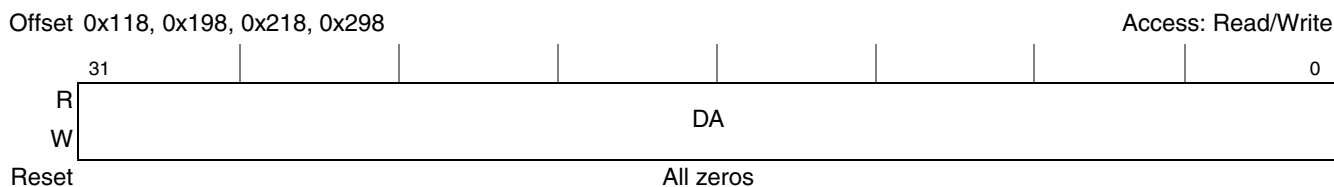


Figure 14-14. DMA Destination Address Register (DMADAR_n)

[Table 14-14](#) describes the DMADAR_n register.

Table 14-14. DMASAR_n Field Descriptions

Bits	Name	Description
31–0	DA	Destination address of DMA transfer. Updated after each DMA write operation.

14.3.8.6 DMA Byte Count Register (DMABCRn)

DMABCRn contains the number of bytes per transfer (maximum transfer size is 64 Mbytes). Figure 14-15 shows the DMABCRn.

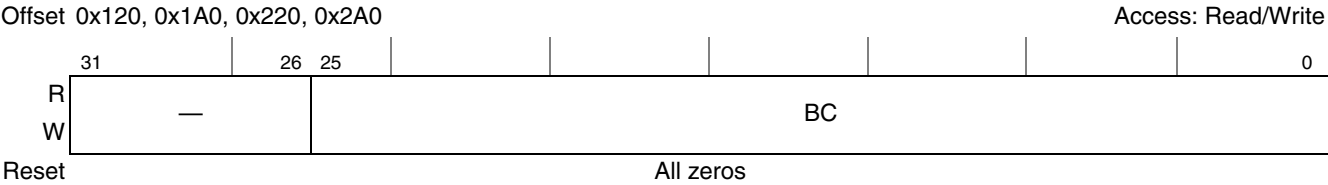


Figure 14-15. DMA Byte Count Register (DMABCRn)

Table 14-15 describes the DMABCRn register.

Table 14-15. DMABCRn 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.

14.3.8.7 DMA Next Descriptor Address Register (DMANDARn)

DMANDARn 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 14-16 shows the DMANDARn.

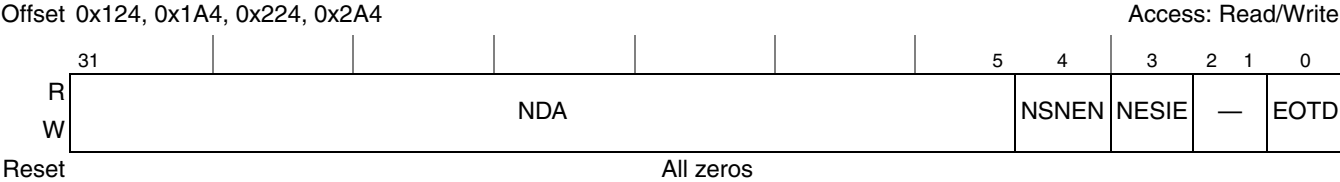


Figure 14-16. DMA Next Descriptor Address Register (DMANDARn)

Table 14-16 describes the DMANDARn register.

Table 14-16. DMANDARn 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.

Table 14-16. DMANDAR_n Field Descriptions (continued)

Bits	Name	Descriptions
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.

14.3.8.8 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 14-17](#) shows the DMAGSR fields.

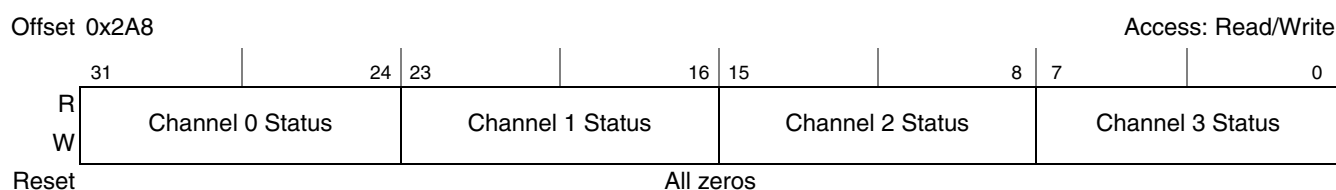


Figure 14-17. DMA General Status Register (DMAGSR)

14.4 Functional Description

14.4.1 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.

14.4.1.1 Messaging Registers (IMR0–IMR1, OMR0–OMR1)

There are two 32-bit inbound message registers (IMR0–IMR1) and two 32-bit outbound message registers (OMR0–OMR1). IMR0 and IMR1 allow 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 $\overline{\text{PCI_INTA}}$ to assert.

The interrupt to the local processor is cleared by writing 1 to the appropriate IMISR bit. The interrupt to PCI ($\overline{\text{PCI_INTA}}$) is cleared by writing 1 to the appropriate OMISR bit.

14.4.1.2 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 PCI_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 (PCI_INTA) is cleared by writing 1 to the appropriate ODR bit.

14.4.2 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 14-18 shows a diagram of the DMA controller in the integrated device.

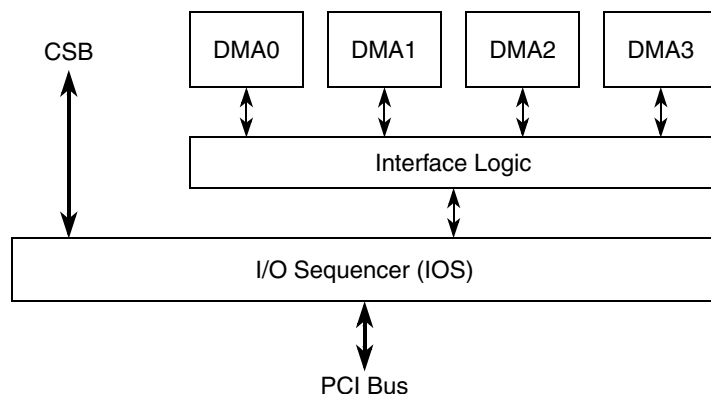


Figure 14-18. DMA Controller Block Diagram

14.4.3 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 14.5.1, “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 14.5.2, “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 misaligned 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. On misaligned addresses, full cache line transfers (32 byte bursting) occur if the source and destination offsets end in the same byte offset. For example, if the destination address is 0x4000_1050 and the source address is 0x9000_2050, the transfer bursts because both end in 0x50. However, if the destination address is 0x4000_1050 and the source is 0x9000_2000, the transfer does not burst because the last byte offset is not the same. On misaligned destination addresses, subtransfers of less than a cache line occur on the initial and final beats of the transfer while full cache lines occur on intermediate beats.

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.

14.4.3.1 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.

14.4.3.2 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.

14.4.4 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 14-17. 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 (DMASAR n).
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 (DMADAR n).
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 (DMANDAR n).
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 (DMABCR n).

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).

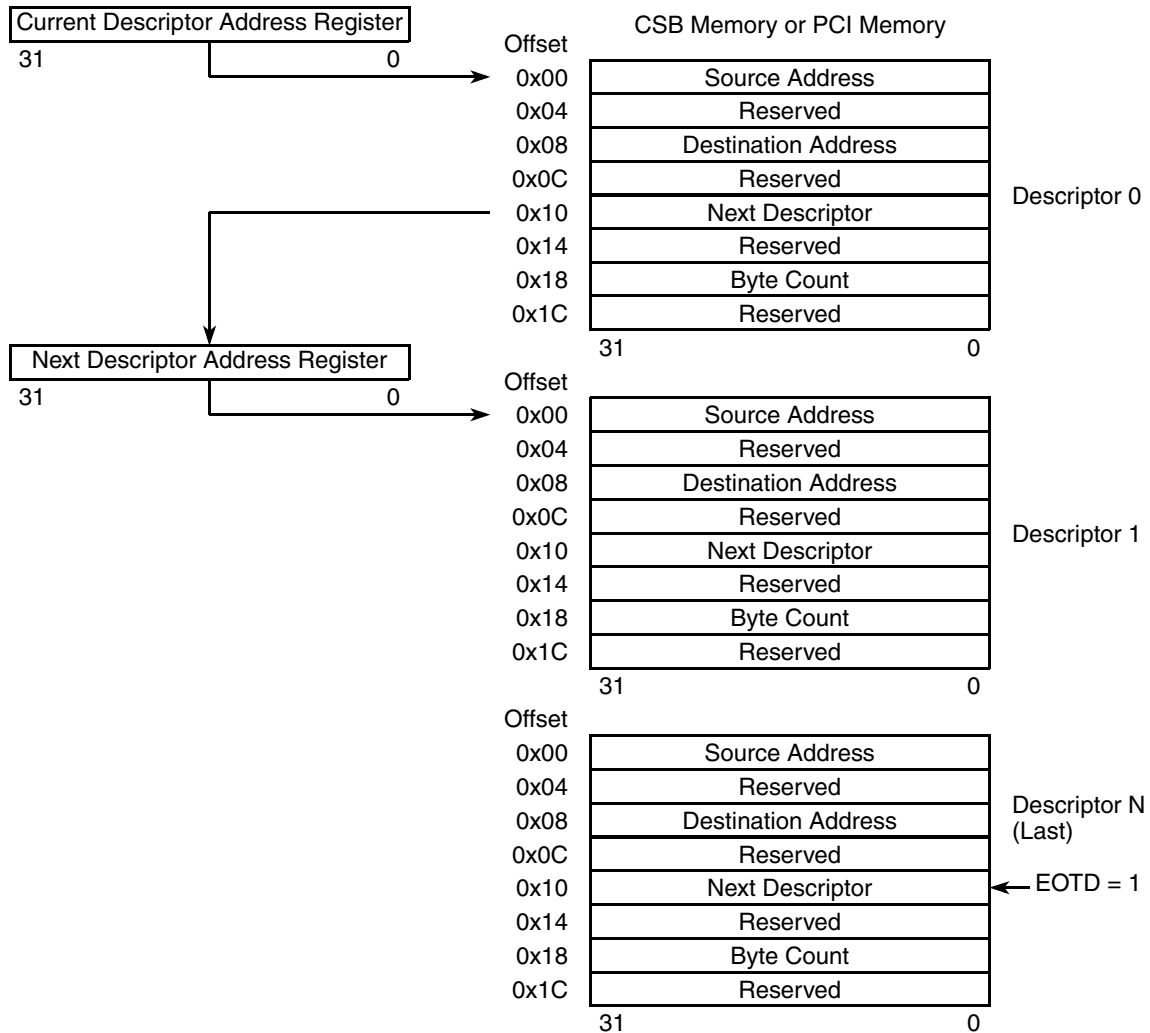


Figure 14-19. DMA Chain of Segment Descriptors

14.4.4.1 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>

```

14.4.4.2 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>

```

14.5 Initialization/Application Information

14.5.1 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.

14.5.2 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 14.4.4, “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.



Chapter 15

FlexCAN

15.1 Introduction

The FlexCAN module is a communication controller implementing the CAN protocol according to the CAN 2.0B protocol specification. A general block diagram is shown in [Figure 15-1](#) that describes the main sub-blocks implemented in the FlexCAN module, including two embedded memories, one for storing message buffers (MB) and another for storing Rx individual mask registers. Support for up to 64 MB is provided. The functions of the sub-modules are described in subsequent sections.

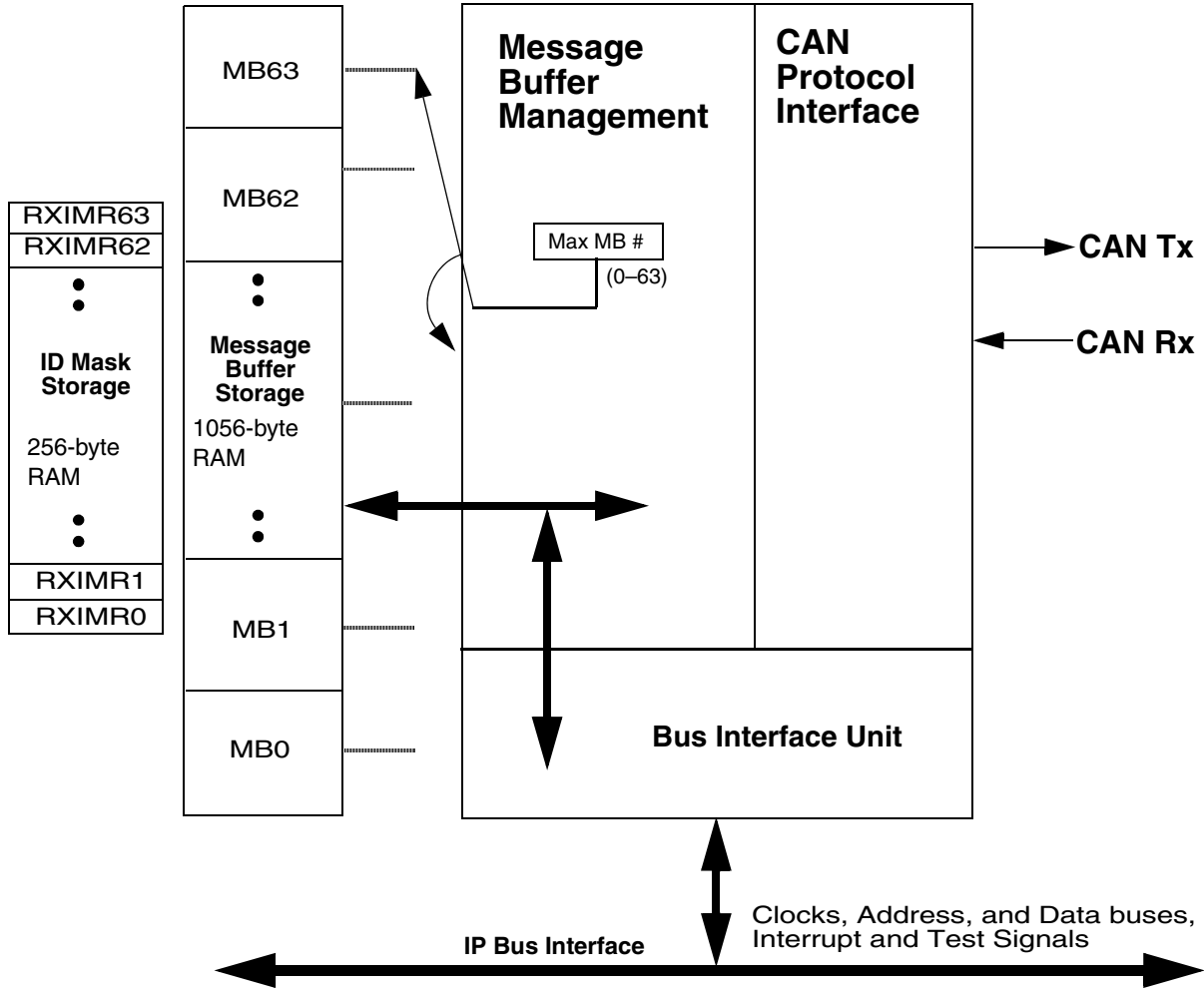


Figure 15-1. FlexCAN Block Diagram

15.1.1 Overview

The CAN protocol is primarily designed to be used as a vehicle serial data bus, meeting the specific requirements of this field: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness, and required bandwidth. The FlexCAN module is a full implementation of the CAN protocol specification, Version 2.0 B that supports both standard and extended message frames. A flexible

number of MB (16, 32, or 64) is also supported. The message buffers are stored in an embedded RAM dedicated to the FlexCAN module. FlexCAN sub-modules handles following operations:

- The CAN protocol interface (CPI) sub-module manages the serial communication on the CAN bus, requesting RAM access for receiving and transmitting message frames, validating received messages, and performing error handling.
- The message buffer management (MBM) sub-module handles MB selection for reception and transmission, arbitration, and ID match algorithms.
- The bus interface unit (BIU) sub-module controls the access to and from the internal interface bus to establish connection to the CPU and to other blocks. Clocks, address and data buses, interrupt outputs, and test signals are also accessed through the BIU.

15.1.2 FlexCAN Module Features

The FlexCAN module includes following features:

- Full implementation of the CAN protocol specification, Version 2.0B
 - Standard data and remote frames
 - Extended data and remote frames
 - Zero to eight bytes data length
 - Programmable bit rate up to 1 Mbps
 - Content-related addressing
- Flexible MB (up to 64) of 0 to 8 bytes data length
- Each MB configurable as Rx or Tx, all supporting standard and extended messages
- Individual Rx mask registers per MB
- Includes 1056 bytes (64 MBs) of RAM used for MB storage
- Includes 256 bytes (64 MBs) of RAM used for individual Rx mask registers
- Full featured Rx FIFO with storage capacity for six frames and internal pointer handling
- Powerful Rx FIFO ID filtering, capable of matching incoming IDs against either 8 extended, 16 standard, or 32 partial (8-bit) IDs, with individual masking capability
- Selectable backwards compatibility with previous FlexCAN version
- Unused MB and Rx mask register space can be used as general purpose RAM space
- Listen only mode capability
- Programmable loop-back mode supporting self-test operation
- Programmable transmission priority scheme: lowest ID, lowest buffer number, or highest priority
- Time stamp based on 16-bit free-running timer
- Global network time, synchronized by a specific message
- Maskable interrupts
- Independent of the transmission medium (an external transceiver is assumed)
- Short latency time due to an arbitration scheme for high-priority messages

15.1.2.1 MPC8309 Specific Features

MPC8309 has four controller area network (FlexCAN) blocks.

- FlexCAN 0 (CAN0)
- FlexCAN 1 (CAN1)
- FlexCAN 2 (CAN2)
- FlexCAN 3 (CAN3)

Each FlexCAN module has the following specific features implemented:

- Embedded memory capable of storing 64 MBs
- FlexCAN bit timing logic can operate with either system clock or external 4–40 MHz oscillator clock.

15.1.3 Modes of Operation

The FlexCAN module has following functional modes:

- **Normal Mode:** In normal mode, the module operates receiving and/or transmitting message frames, errors are handled normally and all the CAN protocol functions are enabled..
- **Freeze Mode:** It is enabled when the FRZ bit in the MCR register is asserted. If enabled, freeze mode is entered when the HALT bit in MCR is set or when debug mode is requested at MCU level. In this mode, no transmission or reception of frames is done and synchronicity to the CAN bus is lost. See [Section 15.4.9.1, “Freeze Mode,”](#) for more information.
- **Listen-Only Mode:** The module enters this mode when the LOM bit in the control register is asserted. In this mode, transmission is disabled, all error counters are frozen, and the module operates in a CAN error passive mode. Only messages acknowledged by another CAN station are received. If FlexCAN detects a message that has not been acknowledged, it flags a BIT0 error (without changing the REC), as if it was trying to acknowledge the message.
- **Loop-Back Mode:** The module enters this mode when the LPB bit in the control register is asserted. In this mode, FlexCAN performs an internal loop back that can be used for self-test operation. The bit stream output of the transmitter is internally fed back to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic ‘1’). FlexCAN behaves as it normally does when transmitting. It treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field to ensure proper reception of its own message. Both transmit and receive interrupts are generated.

The FlexCAN module has following low-power modes:

- **Module Disable Mode:** This low-power mode is entered when the MDIS bit in the MCR register is asserted. When disabled, the module shuts down the clocks to the CPI and MBM sub-modules. Exit from this mode is done by negating the MDIS bit in the MCR register. See [Section 15.4.9.2, “Module Disable Mode,”](#) for more information.
- **Doze Mode:** This low-power mode is entered when the doze bit in MCR is asserted and doze mode is requested at MCU level. When in doze mode, the module shuts down the clocks to the CPI and the MBM sub-modules. Exit from this mode occurs when the doze bit in MCR is negated, when

the MCU is removed from doze mode. See [Section 15.4.9, “Modes of Operation Details,”](#) for more information.

- **Stop Mode:** This low-power mode is entered when stop mode is requested at MCU level. In stop mode, the module puts itself in an inactive state and then informs the CPU that the clocks can be shut down globally. Exit from this mode happens when the stop mode request is removed. See [Section 15.4.9, “Modes of Operation Details,”](#) for more information.

15.2 External Signal Description

15.2.1 Signals Overview

The FlexCAN module has two I/O signals connected to the external MCU pins. These signals are summarized in [Table 15-1](#) and described in more detail in the next subsections.

Table 15-1. FlexCAN Signals

Name	Function/Description	Reset	I/O	Pins
CAN _n _RX	CAN Receive Pin: This pin is the receive pin from the CAN bus transceiver. Dominant state is represented by logic level '0'. Recessive state is represented by logic level '1'.	1	I/O	2
CAN _n _TX	CAN Transmit Pin: This pin is the transmit pin to the CAN bus transceiver. Dominant state is represented by logic level '0'. Recessive state is represented by logic level '1'.	1	I/O	2

15.3 Memory Map/Register Definition

This section describes the registers and data structures in the FlexCAN module. The addresses presented here are relative to the base address.

The address space occupied by FlexCAN has 96 bytes for registers starting at the module base address, followed by MB storage space in embedded RAM starting at address 0x0060, and an extra ID mask storage space in a separate embedded RAM starting at address 0x0880.

The IFLAG2 and IMASK2 registers are considered reserved space when FlexCAN is configured with 16 or 32 MBs. The Rx global mask (RXGMASK), Rx buffer 14 mask (RX14MASK), and the Rx buffer 15 mask (RX15MASK) registers are provided for backwards compatibility, and are not used when the BCC bit in MCR is asserted.

The address ranges 0x0060–0x047F and 0x0880–0x097F are occupied by two separate embedded memories. These two ranges are completely occupied by RAM (1056 and 256 bytes, respectively) only when FlexCAN is configured with 64 MBs. When it is configured with 16 MBs, the memory sizes are 288 and 64 bytes, so the address ranges 0x0180–0x047F and 0x08C0–0x097F are considered reserved space. When it is configured with 32 MBs, the memory sizes are 544 and 128 bytes, so the address ranges 0x0280–0x047F and 0x0900–0x097F are considered reserved space. Furthermore, if the BCC bit in MCR is negated, then the whole Rx individual mask registers address range (0x0880–0x097F) is considered reserved space.

The complete memory map for a FlexCAN module with 64 MBs capability is shown in [Table 15-2](#).

Table 15-2. FlexCAN Memory Map

Offset	Register	Reset	Affected by Hard Reset	Affected by Soft Reset	Section/ Page
CAN1—Block Base Address 0x1_C000 CAN2—Block Base Address 0x1_D000 CAN3—Block Base Address 0x2_9000 CAN4—Block Base Address 0x2_A000					
0x000	Module configuration (MCR)	5980_000F	Yes	Yes	15.3.3.1/15-12
0x004	Control register (CTRL)	All zeros	Yes	No	15.3.3.2/15-15
0x008	Free running timer (TIMER)	All zeros	Yes	Yes	15.3.3.3/15-18
0x00C	Reserved	—	—	—	—
0x010	Rx Global Mask (RXGMASK)	FFFF_FFFF	Yes	No	15.3.3.4/15-19
0x014	Rx Buffer 14 Mask (RX14MASK)	FFFF_FFFF	Yes	No	15.3.3.5/15-19
0x018	Rx Buffer 15 Mask (RX15MASK)	FFFF_FFFF	Yes	No	15.3.3.6/15-20
0x01C	Error Counter Register (ECR)	All zeros	Yes	Yes	15.3.3.7/15-20
0x020	Error and Status Register (ESR)	All zeros	Yes	Yes	15.3.3.8/15-22
0x024	Interrupt Masks 2 (IMASK2)	All zeros	Yes	Yes	15.3.3.9/15-24
0x028	Interrupt Masks 1 (IMASK1)	All zeros	Yes	Yes	15.3.3.10/15-25
0x02C	Interrupt Flags 2 (IFLAG2)	All zeros	Yes	Yes	15.3.3.11/15-25
0x030	Interrupt Flags 1 (IFLAG1)	All zeros	Yes	Yes	15.3.3.12/15-26
0x034–0x05F	Reserved	—	—	—	—
0x060–0x07F	Reserved	—	—	—	—
0x080–0x17F	Message Buffers MB0–MB15	X ¹	No	No	15.3.1/15-7
0x180–0x27F	Message Buffers MB16–MB31	X ¹	No	No	15.3.1/15-7
0x280–0x47F	Message Buffers MB32–MB63	X ¹	No	No	15.3.1/15-7
0x480–087F	Reserved	—	—	—	—
0x880–0x08BF	Rx Individual Mask Registers RXIMR0–RXIMR15	X ¹	No	No	15.3.3.13/15-27
0x8C0–0x08FF	Rx Individual Mask Registers RXIMR16–RXIMR31	X ¹	No	No	15.3.3.13/15-27
0x900–0x097F	Rx Individual Mask Registers RXIMR32–RXIMR63	X ¹	No	No	15.3.3.13/15-27

¹ This register is implemented in internal SRAM, hence no reset value.

The FlexCAN module stores CAN messages for transmission and reception using a message buffer structure. Each individual MB is formed by 16 bytes mapped on memory.

Table 15-3 shows a standard/extended message buffer (MB0) memory map, using 16 bytes total (0x80–0x8F space).

Table 15-3. Message Buffer MB0 Memory Mapping

Offset	MB Field
0x80	Control and Status (C/S)
0x84	Identifier Field
0x88–0x8F	Data Field 0 – Data Field 7 (1 byte each)

15.3.1 Message Buffer Structure

The message buffer structure used by the FlexCAN module is represented in Figure 15-2. Both extended and standard frames (29-bit identifier and 11-bit identifier, respectively) used in the CAN specification (Version 2.0 Part B) are represented.

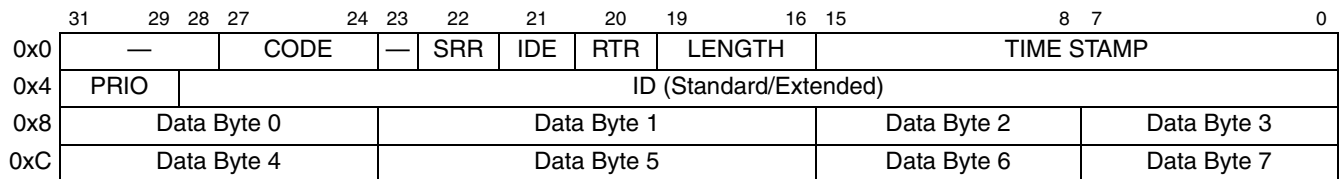


Figure 15-2. Message Buffer Structure

Table 15-4 shows message buffer description.

Table 15-4. Message Buffer Description

Bits	Name	Description
0x0		
31–28	—	Reserved
27–24	CODE	Message Buffer Code This 4-bit field can be accessed (read or write) by the CPU and by the Flexcan module itself as part of the message buffer matching and arbitration process. The encoding is shown in Table 15-5 and Table 15-6. See Section 15.4, “Functional Description,” for additional information.
23	—	Reserved
22	SRR	Substitute Remote Request: Fixed recessive bit, used only in extended format. It must be set to ‘1’ by the user for transmission (Tx Buffers) and is stored with the value received on the CAN bus for Rx receiving buffers. It can be received as either recessive or dominant. If FlexCAN receives this bit as dominant, then it is interpreted as arbitration loss. 1 Recessive value is compulsory for transmission in extended format frames 0 Dominant is not a valid value for transmission in extended format frames
21	IDE	ID Extended Bit: This bit identifies whether the frame format is standard or extended. 1 Frame format is extended 0 Frame format is standard

Table 15-4. Message Buffer Description (continued)

Bits	Name	Description
20	RTR	Remote Transmission Request: This bit is used for requesting transmissions of a data frame. If FlexCAN transmits this bit as '1' (recessive) and receives it as '0' (dominant), it is interpreted as arbitration loss. If this bit is transmitted as '0' (dominant), then if it is received as '1' (recessive), the FlexCAN module treats it as bit error. If the value received matches the value transmitted, it is considered as a successful bit transmission. 1 Indicates the current MB has a remote frame to be transmitted 0 Indicates the current MB has a data frame to be transmitted
19–16	LENGTH	Length of Data in Bytes: This 4-bit field is the length (in bytes) of the Rx or Tx data that is located in offset 0x8 through 0xF of the MB space (see Figure 15-2). In reception, this field is written by the FlexCAN module, copied from the DLC (Data Length Code) field of the received frame. In transmission, this field is written by the CPU and corresponds to the DLC field value of the frame to be transmitted. When RTR=1, the Frame to be transmitted is a Remote Frame and does not include the data field, regardless of the Length field.
15–0	TIME STAMP	Free-Running Counter Time Stamp: This 16-bit field is a copy of the Free-Running Timer, captured for Tx and Rx frames at the time when the beginning of the Identifier field appears on the CAN bus.
0x4		
31–29	PRIO	Local priority: This 3-bit field is only used when LPRIO_EN bit is set in MCR and it only makes sense for Tx buffers. These bits are not transmitted. They are appended to the regular ID to define the transmission priority. See Section 15.4.3, "Arbitration process."
28–0	ID	Frame Identifier: In standard frame format, only the 11 most significant bits (28 to 18) are used for frame identification in both receive and transmit cases. The 18 least significant bits are ignored. In extended frame format, all bits are used for frame identification in both receive and transmit cases.
0x8		
31–24	Data byte 0	Data Field: Up to 8 bytes can be used for a data frame. For Rx frames, the data is stored as it is received from the CAN bus. For Tx frames, the CPU prepares the data field to be transmitted within the frame.
23–16	Data byte 1	
15–8	Data byte 2	
7–0	Data byte 3	
0xC		
31–24	Data byte 4	Data Field: Up to 8 bytes can be used for a data frame. For Rx frames, the data is stored as it is received from the CAN bus. For Tx frames, the CPU prepares the data field to be transmitted within the frame.
23–16	Data byte 5	
15–8	Data byte 6	
7–0	Data byte 7	

The encoding of CODE field is shown in [Table 15-5](#) and [Table 15-6](#)

Table 15-5. Message Buffer Code for Rx buffers

Rx Code before Rx New Frame	Description	Rx Code after Rx New Frame	Comment
0000	INACTIVE: MB is not Active.	—	MB does not participate in the matching process.
0100	EMPTY: MB is active and empty.	0010	MB participates in the matching process. When a frame is received successfully, the code is automatically updated to FULL.
0010	FULL: MB is full.	0010	The act of reading the C/S word followed by unlocking the MB does not make the code return to EMPTY. It remains FULL. If a new frame is written to the MB after the C/S word was read and the MB was unlocked, the code still remains FULL.
		0110	If the MB is FULL and a new frame is overwritten to this MB before the CPU had time to read it, the code is automatically updated to OVERRUN. Refer to Section 15.4.5, “Matching Process,” for details about overrun behavior.
0110	OVERRUN: A frame was overwritten into a full buffer.	0010	If the code indicates OVERRUN but the CPU reads the C/S word and then unlocks the MB, when a new frame is written to the MB the code returns to FULL.
		0110	If the code already indicates OVERRUN, and yet another new frame must be written, the MB is overwritten again, and the code remains OVERRUN. Refer to Section 15.4.5, “Matching Process,” for details about overrun behavior.
0XY1 ¹	BUSY: Flexcan is updating the contents of the MB. The CPU must not access the MB.	0010	An EMPTY buffer was written with a new frame (XY was 01).
		0110	A FULL/OVERRUN buffer was overwritten (XY was 11).

¹ Note that for Tx MBs (see [Table 15-6](#)), the BUSY bit should be ignored upon read, except when AEN bit is set in the MCR register.

Table 15-6. Message Buffer Code for Tx buffers

RTR	Initial Tx Code	Code after Successful Transmission	Description
X	1000	—	INACTIVE: MB does not participate in the arbitration process.
X	1001	—	ABORT: MB was configured as Tx and CPU aborted the transmission. This code is only valid when AEN bit in MCR is asserted. MB does not participate in the arbitration process.
0	1100	1000	Transmit data frame unconditionally once. After transmission, the MB automatically returns to the INACTIVE state.
1	1100	0100	Transmit remote frame unconditionally once. After transmission, the MB automatically becomes an Rx MB with the same ID.
0	1010	1010	Transmit a data frame whenever a remote request frame with the same ID is received. This MB participates simultaneously in both the matching and arbitration processes. The matching process compares the ID of the incoming remote request frame with the ID of the MB. If a match occurs, this MB is allowed to participate in the current arbitration process and the Code field is automatically updated to ‘1110’ to allow the MB to participate in future arbitration runs. When the frame is eventually transmitted successfully, the Code automatically returns to ‘1010’ to restart the process again.

Table 15-6. Message Buffer Code for Tx buffers (continued)

RTR	Initial Tx Code	Code after Successful Transmission	Description
0	1110	1010	This is an intermediate code that is automatically written to the MB by the MBM as a result of match to a remote request frame. The data frame is transmitted unconditionally once and then the code is automatically returned to '1010'. The CPU can also write this code with the same effect.

15.3.2 Rx FIFO Structure

When the FEN bit is set in the MCR, the memory area from 0x80 to 0xFC (that is normally occupied by MBs 0–7) is used by the reception FIFO engine. [Figure 15-3](#) shows the Rx FIFO data structure. The region 0x80–0x8C contains an MB structure that is the port through which the CPU reads data from the FIFO (the oldest frame received and not read yet). The region 0x90–0xDC is reserved for internal use of the FIFO engine. The region 0xE0–0xFC contains an 8-entry ID table that specifies filtering criteria for accepting frames into the FIFO. [Figure 15-3](#) shows the three different formats that the elements of the ID table can assume, depending on the IDAM field of the MCR. Note that all elements of the table must have the same format. See [Section 15.4.7, “Rx FIFO,”](#) for more information.

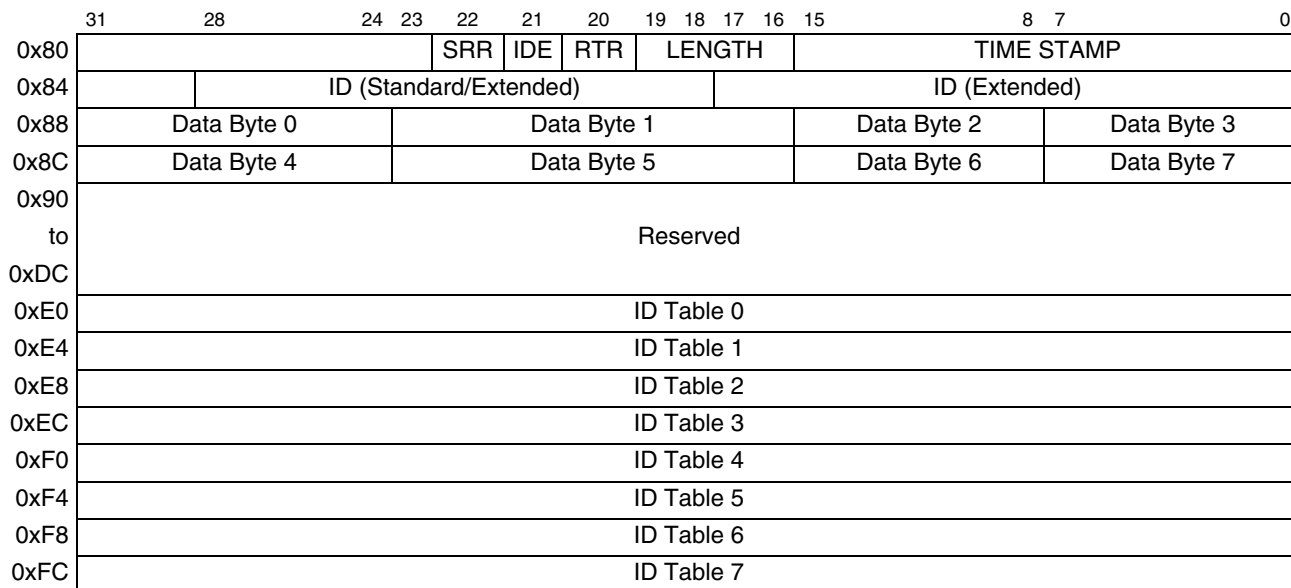


Figure 15-3. Rx FIFO Structure

NOTE

Note that field description for SRR, IDE, RTD, LENGTH, TIME STAMP, ID, and Data is same as that in message buffer structure shown in [Table 15-4](#).

Figure 15-4 shows ID table structure.

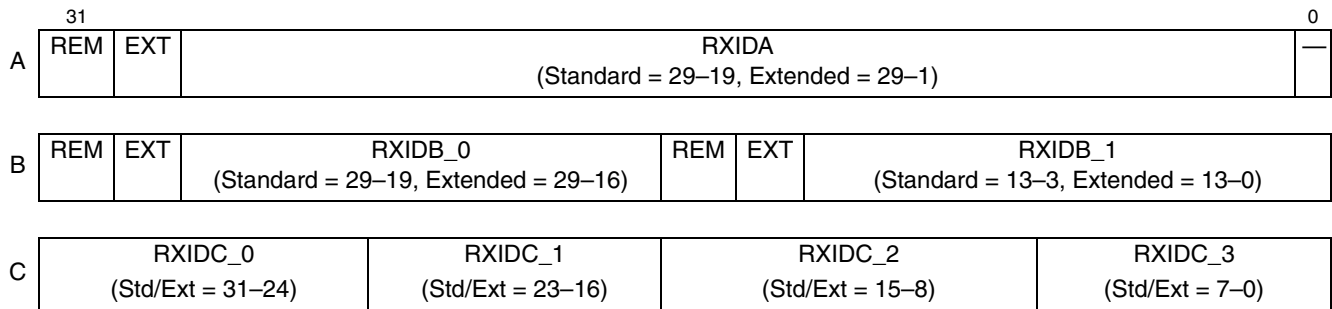


Figure 15-4. ID Table (0–7)

Table 15-7 shows the Table field description.

Table 15-7. ID Table (0–7) Field Description

Bits	Name	Description
31 Format A, Format B	REM	Remote Frame: This bit specifies if Remote Frames are accepted into the FIFO if they match the target ID 1 Remote frames can be accepted and data frames are rejected 0 Remote frames are rejected and data frames can be accepted
30 Format A, Format B	EXT	Extended Frame Specifies whether extended or standard frames are accepted into the FIFO if they match the target ID. 1 Extended frames can be accepted and standard frames are rejected 0 Extended frames are rejected and standard frames can be accepted
Format A		
29–1	RXIDA	Rx Frame Identifier: Specifies an ID to be used as acceptance criteria for the FIFO. In the standard frame format, only the 11 most significant bits (29 to 19) are used for frame identification. In the extended frame format, all bits are used.
0	—	Reserved
Format B		
29–16 13–0	RXIDB_0, RXIDB_1	Rx Frame Identifier: Specifies an ID to be used as acceptance criteria for the FIFO. In the standard frame format, the 11 most significant bits (a full standard ID) (29 to 19 and 13 to 3) are used for frame identification. In the extended frame format, all 14 bits of the field are compared to the 14 most significant bits of the received ID.
15	REM	Remote Frame: This bit specifies if Remote Frames are accepted into the FIFO if they match the target ID. 1 Remote frames can be accepted and data frames are rejected 0 Remote frames are rejected and data frames can be accepted
14	EXT	Extended Frame: Specifies whether extended or standard frames are accepted into the FIFO if they match the target ID. 1 Extended frames can be accepted and standard frames are rejected 0 Extended frames are rejected and standard frames can be accepted

Table 15-7. ID Table (0–7) Field Description (continued)

Bits	Name	Description
Format C		
31–0	RXIDC_0, RXIDC_1, RXIDC_2, RXIDC_3	Rx Frame Identifier: Specifies an ID to be used as acceptance criteria for the FIFO. In both standard and extended frame formats, all 8 bits of the field are compared to the 8 most significant bits of the received ID.

15.3.3 Register Descriptions

The FlexCAN registers described in this section are in ascending address order.

15.3.3.1 Module Configuration Register (MCR)

This register defines global system configurations, such as the module operation mode (low-power) and maximum message buffer configuration. This register can be accessed at any time; however, some fields must be changed only during freeze mode.

Figure 15-5 shows module configuration register.

Offset 0x000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	MDIS	FRZ	FEN	HALT	NOT_RDY	—	SOFT_RST	FRZ_ACK	—	—	WRN_EN	—	—	—	SRX_DIS	BCC
W					—			—								
Reset	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	—	—	LPRIO_EN	AEN	—	—	IDAM	—	—	—	—	—	—	—	—	—
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 15-5. Module Configuration Register (MCR)

Table 15-8 shows the module configuration register description.

Table 15-8. Module Configuration Register Description

Bits	Name	Description
31	MDIS	<p>Module Disable: This bit controls whether FlexCAN is enabled or not. When disabled, FlexCAN shuts down the clocks to the CPI and MBM sub-modules. This is the only bit in MCR not affected by soft reset. See Section 15.4.9.2, “Module Disable Mode,” for more information.</p> <p>0 Enable the FlexCAN module 1 Disable the FlexCAN module</p>
30	FRZ	<p>Freeze Enable: The FRZ bit specifies the FlexCAN behavior when the HALT bit in the MCR Register is set or when Debug Mode is requested at MCU level. When FRZ is asserted, FlexCAN is enabled to enter freeze mode. Negation of this bit field causes FlexCAN to exit from freeze mode.</p> <p>1 Enabled to enter freeze mode 0 Not enabled to enter freeze mode</p>
29	FEN	<p>FIFO Enable: This bit controls whether the FIFO feature is enabled or not. When FEN is set, MBs 0–7 cannot be used for normal reception and transmission because the corresponding memory region (0x80–0xFF) is used by the FIFO engine. See Section 15.3.2, “Rx FIFO Structure” and Section 15.4.7, “Rx FIFO,” for more information. This bit must be written in freeze mode only.</p> <p>1 FIFO enabled 0 FIFO not enabled</p>
28	HALT	<p>Halt FlexCAN: Assertion of this bit puts the FlexCAN module into freeze mode. The CPU should clear it after initializing the message buffers and control register. No reception or transmission is performed by FlexCAN before this bit is cleared. While in freeze mode, the CPU has write access to the error counter register, which is otherwise read-only. Freeze Mode cannot be entered while FlexCAN is in any of the low-power modes. See Section 15.4.9.1, “Freeze Mode,” for more information.</p> <p>1 Enters freeze mode if the FRZ bit is asserted. 0 No Freeze Mode request.</p>
27	NOT_RDY	<p>FlexCAN Not Ready: This read-only bit indicates that FlexCAN is either in disable mode, doze mode, stop mode, or freeze mode. It is negated once FlexCAN has exited these modes.</p> <p>1 FlexCAN module is either in disable mode, doze mode, stop mode, or freeze mode 0 FlexCAN module is either in normal mode, listen-only mode, or loop-back mode</p>
26	—	
25	SOFT_RST	<p>Soft Reset: When this bit is asserted, FlexCAN resets its internal state machines and some of the memory-mapped registers. The following registers are reset: MCR (except the MDIS bit), TIMER, ECR, ESR, IMASK1, IMASK2, IFLAG1, IFLAG2. Configuration registers that control the interface to the CAN bus are not affected by soft reset. The following registers are unaffected:</p> <ul style="list-style-type: none"> • CTRL • RXIMR0–RXIMR63 • RXGMASK, RX14MASK, RX15MASK • all Message Buffers <p>The SOFT_RST bit can be asserted directly by the CPU when it writes to the MCR Register, but it is also asserted when global soft reset is requested at MCU level. Since soft reset is synchronous and has to follow a request/acknowledge procedure across clock domains, it may take some time to fully propagate its effect. The SOFT_RST bit remains asserted while reset is pending, and is automatically negated when reset completes. Therefore, software can poll this bit to know when the soft reset has completed.</p> <p>Soft reset cannot be applied while clocks are shut down in any of the low-power modes. The module should be first removed from low-power mode, and then soft reset can be applied.</p> <p>1 Resets the registers marked as “affected by soft reset” in Table 15-2 0 No reset request</p>

Table 15-8. Module Configuration Register Description (continued)

Bits	Name	Description
24	FRZ_ACK	<p>Freeze Mode Acknowledge: This read-only bit indicates that FlexCAN is in freeze mode and its prescaler is stopped. The freeze mode request cannot be granted until current transmission or reception processes have finished. Therefore, the software can poll the FRZ_ACK bit to know when FlexCAN has actually entered freeze mode. If freeze mode request is negated, then this bit is negated once the FlexCAN prescaler is running again. If freeze mode is requested while FlexCAN is in any of the low-power modes, then the FRZ_ACK bit is only set when the low-power mode is exited. See Section 15.4.9.1, “Freeze Mode,” for more information.</p> <p>1 FlexCAN in freeze mode, prescaler stopped 0 FlexCAN not in freeze mode, prescaler running</p>
23	—	Reserved
22	—	Reserved
21	WRN_EN	<p>Warning Interrupt Enable: When asserted, this bit enables the generation of the TWRN_INT and RWRN_INT flags in the Error and Status Register. If WRN_EN is negated, the TWRN_INT and RWRN_INT flags are always zero, independent of the values of the error counters, and no warning interrupt is ever generated. This bit must be written in freeze mode only.</p> <p>1 TWRN_INT and RWRN_INT bits are set when the respective error counter transition from <96 to ≥96. 0 TWRN_INT and RWRN_INT bits are zero, independent of the values in the error counters.</p>
20–18	—	Reserved
17	SRX_DIS	<p>Self-Reception Disable: This bit defines whether FlexCAN is allowed to receive frames transmitted by itself. If this bit is asserted, frames transmitted by the module is not stored in any MB, regardless if the MB is programmed with an ID that matches the transmitted frame, and no interrupt flag or interrupt signal is generated due to the frame reception. This bit must be written in freeze mode only.</p> <p>1 Self-reception disabled 0 Self-reception enabled</p>
16	BCC	<p>Backwards Compatibility Configuration: This bit is provided to support backwards compatibility with previous FlexCAN versions. When this bit is negated, the following configuration is applied:</p> <ul style="list-style-type: none"> For MCUs supporting individual Rx ID masking, this feature is disabled. Instead of individual ID masking per MB, FlexCAN uses its previous masking scheme with RXGMASK, RX14MASK, and RX15MASK. The reception queue feature is disabled. Upon receiving a message, if the first MB with a matching ID that is found is still occupied by a previous unread message, FlexCAN does not look for another matching MB. It overrides this MB with the new message and set the CODE field to '0110' (overrun). <p>Upon reset this bit is negated, allowing legacy software to work without modification. This bit must be written in freeze mode only.</p> <p>1 Individual Rx masking and queue feature are enabled. 0 Individual Rx masking and queue feature are disabled.</p>
15–14	—	Reserved
13	LPRIO_EN	<p>Local Priority Enable: This bit is provided for backwards compatibility reasons. It controls whether the local priority feature is enabled or not. It is used to extend the ID used during the arbitration process. With this extended ID concept, the arbitration process is done based on the full 32-bit word, but the actual transmitted ID still has 11-bit for standard frames and 29-bit for extended frames. This bit must be written in freeze mode only.</p> <p>1 Local Priority enabled 0 Local Priority disabled</p>

Table 15-8. Module Configuration Register Description (continued)

Bits	Name	Description															
12	AEN	Abort Enable: This bit is supplied for backwards compatibility reasons. When asserted, it enables the Tx abort feature. This feature guarantees a safe procedure for aborting a pending transmission so that no frame is sent in the CAN bus without notification. This bit must be written in freeze mode only. 1 Abort enabled 0 Abort disabled															
11–10	—	Reserved															
9–8	IDAM	ID Acceptance Mode: This 2-bit field identifies the format of the elements of the Rx FIFO filter table, as shown below. Note that all elements of the table are configured at the same time by this field (they are all the same format). See Section 15.3.2, “Rx FIFO Structure . This bit must be written in freeze mode only. IDAM Coding <table border="1"> <thead> <tr> <th>IDAM</th> <th>Format</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>0b00</td> <td>A</td> <td>One full ID (standard or extended) per filter element.</td> </tr> <tr> <td>0b01</td> <td>B</td> <td>Two full standard IDs or two partial 14-bit extended IDs per filter element.</td> </tr> <tr> <td>0b10</td> <td>C</td> <td>Four partial 8-bit IDs (standard or extended) per filter element.</td> </tr> <tr> <td>0b11</td> <td>D</td> <td>All frames rejected.</td> </tr> </tbody> </table>	IDAM	Format	Description	0b00	A	One full ID (standard or extended) per filter element.	0b01	B	Two full standard IDs or two partial 14-bit extended IDs per filter element.	0b10	C	Four partial 8-bit IDs (standard or extended) per filter element.	0b11	D	All frames rejected.
IDAM	Format	Description															
0b00	A	One full ID (standard or extended) per filter element.															
0b01	B	Two full standard IDs or two partial 14-bit extended IDs per filter element.															
0b10	C	Four partial 8-bit IDs (standard or extended) per filter element.															
0b11	D	All frames rejected.															
7–6	—	Reserved															
5–0	MAXMB	Maximum Number of Message Buffers: This 6-bit field defines the maximum number of message buffers that take part in the matching and arbitration processes. The reset value (0x0F) is equivalent to 16 MB configuration. This field must be changed only while the module is in freeze mode. Maximum MBs in use = MAXMB + 1. Note: MAXMB must be programmed with a value smaller or equal to the number of available Message Buffers, otherwise FlexCAN can transmit and receive wrong messages.															

15.3.3.2 Control Register (CTRL)

This register is defined for specific FlexCAN control features related to the CAN bus, such as bit-rate, programmable sampling point within an Rx bit, loop back mode, listen only mode, bus-off recovery behavior, and interrupt enabling (bus-off, error, and warning). It also determines the division factor for the clock prescaler. This register can be accessed at any time; however, some fields must be changed only during either disable mode or freeze mode. [Figure 15-6](#) shows the control register.

Offset 0x004

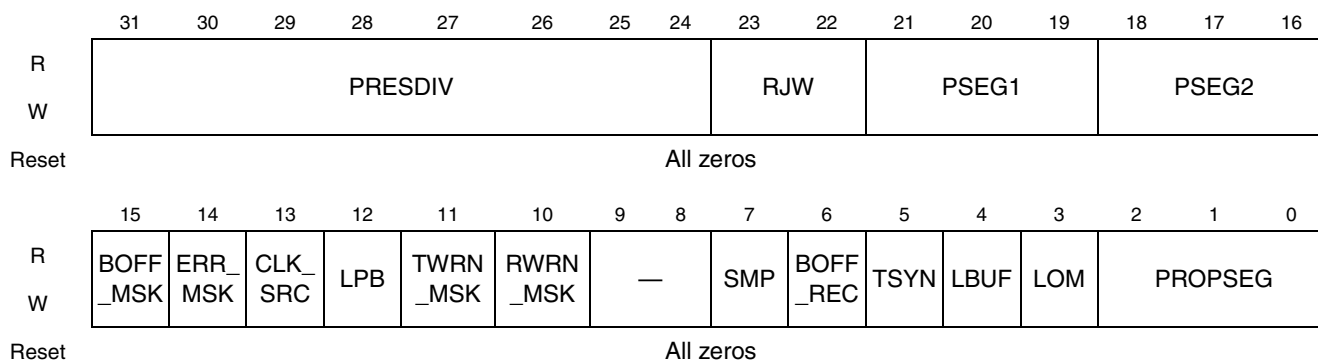


Figure 15-6. Control Register (CTRL)

Table 15-9 shows fields descriptions of control register.

Table 15-9. Control Register Description

Bits	Name	Description
31–24	PRESDIV	Prescaler Division Factor: This 8-bit field defines the ratio between the CPI clock frequency and the Serial Clock (Sclock) frequency. The Sclock period defines the time quantum of the CAN protocol. For the reset value, the Sclock frequency is equal to the CPI clock frequency. The maximum value of this register is 0xFF that gives a minimum Sclock frequency equal to the CPI clock frequency divided by 256. For more information, refer to Section 15.4.8.4, “Protocol Timing.” This bit must be written in freeze mode only. Sclock frequency = CPI clock frequency/(PRESDIV + 1)
23–22	RJW	Resync Jump Width: This 2-bit field defines the maximum number of time quanta ¹ that a bit time can be changed by one re-synchronization. The valid programmable values are 0–3. This bit must be written in freeze mode only. Resync Jump Width = RJW + 1.
21–19	PSEG1	Phase Segment 1: This 3-bit field defines the length of phase buffer segment 1 in the bit time. The valid programmable values are 0–7. This bit must be written in freeze mode only. Phase Buffer Segment 1=(PSEG1 + 1)×Time-Quanta.
18–16	PSEG2	Phase Segment 2: This 3-bit field defines the length of phase buffer segment 2 in the bit time. The valid programmable values are 1–7. This bit must be written in freeze mode only. Phase Buffer Segment 2 = (PSEG2 + 1) x Time-Quanta.
15	BOFF_MSK	Bus-Off Mask: This bit provides a mask for the bus-off Interrupt. 1 Bus-off interrupt enabled 0 Bus-off interrupt disabled
14	ERR_MSK	Error Mask: This bit provides a mask for the Error Interrupt. 1 Error interrupt enabled 0 Error interrupt disabled
13	CLK_SRC	CAN Engine Clock Source: This bit selects the peripheral clock . PLLclock is fed to the prescaler to generate the Serial Clock (Sclock). See Section 15.4.8.4, “Protocol Timing,” for more information. 1 The CAN engine clock source is the bus clock (platform clock) Note: 0 bit is not supported as there is no oscillator clock (selected by value 0).

Table 15-9. Control Register Description (continued)

Bits	Name	Description
12	LPB	<p>Loop Back: This bit configures FlexCAN to operate in loop-back Mode. In this mode, FlexCAN performs an internal loop back that can be used for self-test operation. The bit stream output of the transmitter is fed back internally to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic '1'). FlexCAN behaves as it normally does when transmitting, and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field, generating an internal acknowledge bit to ensure proper reception of its own message. Both transmit and receive interrupts are generated. This bit must be written in freeze mode only.</p> <p>1 Loop Back enabled 0 Loop Back disabled</p>
11	TWRN_MSK	<p>Tx Warning Interrupt Mask: This bit provides a mask for the Tx Warning Interrupt associated with the TWRN_INT flag in the Error and Status Register. This bit has no effect if the WRN_EN bit in MCR is negated and it is read as zero when WRN_EN is negated.</p> <p>1 Tx Warning Interrupt enabled 0 Tx Warning Interrupt disabled</p>
10	RWRN_MSK	<p>Rx Warning Interrupt Mask: This bit provides a mask for the Rx Warning Interrupt associated with the RWRN_INT flag in the Error and Status Register. This bit has no effect if the WRN_EN bit in MCR is negated and it is read as zero when WRN_EN is negated.</p> <p>1 Rx Warning Interrupt enabled 0 Rx Warning Interrupt disabled</p>
9–8	—	Reserved
7	SMP	<p>Sampling Mode: This bit defines the sampling mode of CAN bits at the Rx input. This bit must be written in freeze mode only.</p> <p>1 Three samples are used to determine the value of the received bit: the regular one (sample point) and 2 preceding samples, a majority rule is used 0 Just one sample is used to determine the bit value</p>
6	BOFF_REC	<p>Bus-off Recovery Mode: This bit defines how FlexCAN recovers from bus-off state. If this bit is negated, automatic recovering from bus-off state occurs according to the CAN Specification 2.0B. If the bit is asserted, automatic recovering from bus-off is disabled and the module remains in bus-off state until the bit is negated by the user. If the negation occurs before 128 sequences of 11 recessive bits are detected on the CAN bus, then bus-off recovery happens as if the BOFF_REC bit had never been asserted. If the negation occurs after 128 sequences of 11 recessive bits occurred, then FlexCAN re-synchronizes to the bus by waiting for 11 recessive bits before joining the bus. After negation, the BOFF_REC bit can be re-asserted again during bus-off, but it is only effective to the next time the module enters bus-off. If BOFF_REC was negated when the module entered bus-off, asserting it during bus-off may not be effective for the current bus-off recovery.</p> <p>1 Automatic recovering from bus-off state disabled 0 Automatic recovering from bus-off state enabled, according to CAN Spec 2.0 part B</p>
5	TSYN	<p>Timer Sync Mode: This bit enables a mechanism that resets the free-running timer each time a message is received in Message Buffer 0. This feature provides means to synchronize multiple FlexCAN stations with a special "SYNC" message (global network time). If the FEN bit in MCR is set (FIFO enabled), MB8 is used for timer synchronization instead of MB0. This bit must be written in freeze mode only.</p> <p>1 Timer Sync feature enabled 0 Timer Sync feature disabled</p>

Table 15-9. Control Register Description (continued)

Bits	Name	Description
4	LBUF	Lowest Buffer Transmitted First: This bit defines the ordering mechanism for Message Buffer transmission. When asserted, the LPRIO_EN bit does not affect the priority arbitration. This bit must be written in freeze mode only. 1 Lowest number buffer is transmitted first 0 Buffer with highest priority is transmitted first
3	LOM	Listen-Only Mode: This bit configures FlexCAN to operate in Listen Only Mode. In this mode, transmission is disabled, all error counters are frozen, and the module operates in a CAN Error Passive mode. Only messages acknowledged by another CAN station is received. If FlexCAN detects a message that has not been acknowledged, it is flag a BIT0 error (without changing the REC), as if it was trying to acknowledge the message. This bit must be written in freeze mode only. 1 FlexCAN module operates in Listen Only Mode 0 Listen Only Mode is deactivated
2–0	PROPSEG	Propagation Segment: This 3-bit field defines the length of the Propagation Segment in the bit time. The valid programmable values are 0–7. This bit must be written in freeze mode only. Propagation Segment Time = (PROPSEG + 1) * Time-Quanta. Time-Quantum = one Sclock period.

¹ One time quantum is equal to the Sclock period.

15.3.3.3 Free Running Timer Register (TIMER)

This register represents a 16-bit free running counter that can be read and written by the CPU. The timer starts from 0x0000 after Reset, counts linearly to 0xFFFF, and wraps around.

The timer is clocked by the FlexCAN bit-clock (that defines the baud rate on the CAN bus). During a message transmission/reception, it increments by one for each bit that is received or transmitted. When there is no message on the bus, it counts using the previously programmed baud rate. During freeze mode, the timer is not incremented.

The timer value is captured at the beginning of the identifier field of any frame on the CAN bus. This captured value is written into the Time Stamp entry in a message buffer after a successful reception or transmission of a message.

Writing to the timer is an indirect operation. The data is first written to an auxiliary register and then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user except for the fact that the data takes some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

Offset 0x008

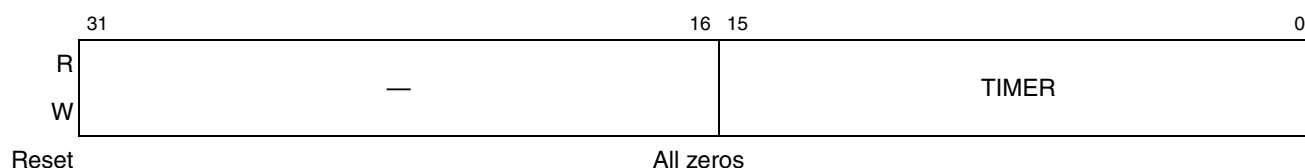


Figure 15-7. Free Running Timer (TIMER)

Table 15-10 shows the free running timer register description

Table 15-10. Free Running Timer (TIMER) Register Description

Bits	Name	Description
31–16	—	Reserved
15–0	TIMER	

15.3.3.4 Rx Global Mask (RXGMASK)

This register is provided for legacy support and for low-cost MCUs that do not have the individual masking per Message Buffer feature. For MCUs supporting individual masks per MB, setting the BCC bit in MCR causes the RXGMASK Register to have no effect on the module operation. For MCUs not supporting individual masks per MB, this register is always effective.

RXGMASK is used as acceptance mask for all Rx MBs, excluding MBs 14–15 that have individual mask registers. When the FEN bit in MCR is set (FIFO enabled), the RXGMASK also applies to all elements of the ID filter table except elements 6–7 that have individual masks.

Refer to [Section 15.4.7, "Rx FIFO"](#) for important details on usage of RXGMASK on filtering process for Rx FIFO.

The contents of this register must be programmed while the module is in freeze mode, and must not be modified when the module is transmitting or receiving frames.

[Figure 15-8](#) shows Rx global mask register (RXGMASK).

Offset 0x010


Figure 15-8. Rx Global Mask Register (RXGMASK)

[Table 15-11](#) shows Rx global mask register description.

Table 15-11. Rx Global Mask Register Description

Bits	Name	Description
31–0	MI31–MI0	Mask Bits: For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 1 The corresponding bit in the filter is checked against the one received 0 The corresponding bit in the filter is “don’t care”

15.3.3.5 Rx 14 Mask Register (RX14MASK)

This register is provided for legacy support and for low-cost MCUs that do not have the individual masking per message buffer feature. For MCUs supporting individual masks per MB, setting the BCC bit in MCR causes the RX14MASK register to have no effect on the module operation.

RX14MASK is used as acceptance mask for the identifier in message buffer 14. When the FEN bit in MCR is set (FIFO enabled), the RXG14MASK also applies to element 6 of the ID filter table. Refer to [Section 15.4.7, "Rx FIFO"](#) for important details on usage of RX14MASK on filtering process for Rx FIFO.

The contents of this register must be programmed while the module is in freeze mode, and must not be modified when the module is transmitting or receiving frames. This register has the same structure as the Rx global mask register shown in [Figure 15-9](#).

Offset 0x014

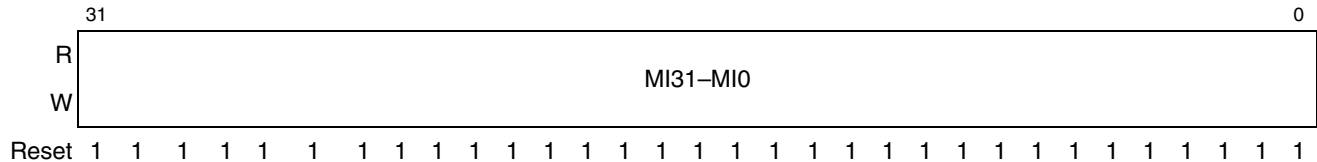


Figure 15-9. Rx 14 Mask Register

[Table 15-11](#) shows Rx 14 mask register description.

15.3.3.6 Rx 15 Mask Register (RX15MASK)

This register is provided for legacy support and for low-cost MCUs that do not have the individual masking per Message Buffer feature. For MCUs supporting individual masks per MB, setting the BCC bit in MCR causes the RX15MASK register to have no effect on the module operation.

When the BCC bit is negated, RX15MASK is used as acceptance mask for the identifier in message buffer 15. When the FEN bit in MCR is set (FIFO enabled), the RXG15MASK also applies to element 7 of the ID filter table. Refer to [Section 15.4.7, "Rx FIFO"](#) for important details on usage of RX15MASK on filtering process for Rx FIFO.

The contents of this register must be programmed while the module is in freeze mode, and must not be modified when the module is transmitting or receiving frames. This register has the same structure as the Rx global mask register.

Offset 0x018



Figure 15-10. Rx 15 Mask Register

[Table 15-11](#) shows Rx 15 mask register description.

15.3.3.7 Error Counter Register (ECR)

This register has following 8-bit fields reflecting the value of two FlexCAN error counters:

- Transmit Error Counter (Tx_Err_Counter field) and
- Receive Error Counter (Rx_Err_Counter field).

The rules for increasing and decreasing these counters are described in the CAN protocol and are completely implemented in the FlexCAN module. Both counters are read-only except in freeze mode, where they can be written by the CPU.

Writing to the error counter register (ECR) while in freeze mode is an indirect operation. The data is first written to an auxiliary register and then an internal request/acknowledge procedure across clock domains is executed. [Figure 15-11](#) shows error counter register.

Offset 0x01C

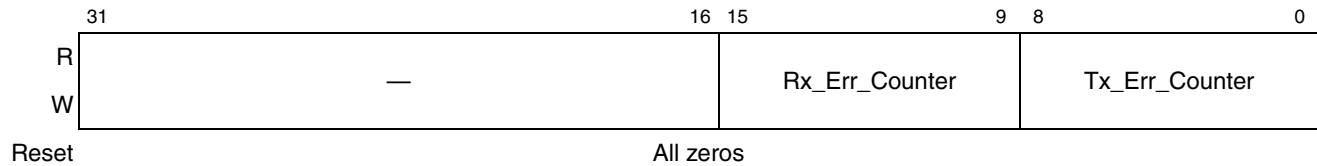


Figure 15-11. Error Counter Register (ECR)

FlexCAN responds to any bus state as described in the protocol, for example, transmit ‘error active’ or ‘error passive’ flag, delay its transmission start time (error passive) and avoid any influence on the bus when in ‘bus-off’ state.

Table 15-12. Error Counter Register Description

Bits	Name	Description
31–16	—	Reserved
15–8	Rx_Err_Counter	Receive Error Counter
9–0	Tx_Err_Counter	Transmitt Error Counter

The following are the basic rules for FlexCAN bus state transitions.

- If the value of Tx_Err_Counter or Rx_Err_Counter increases to be greater than or equal to 128, the FLT_CONF field in the error and status register is updated to reflect ‘error passive’ state.
- If the FlexCAN state is ‘error passive’, and either Tx_Err_Counter or Rx_Err_Counter decrements to a value less than or equal to 127 while the other already satisfies this condition, the FLT_CONF field in the error and status register is updated to reflect ‘error active’ state.
- If the value of Tx_Err_Counter increases to be greater than 255, the FLT_CONF field in the Error and status register is updated to reflect ‘bus-off’ state, and an interrupt may be issued. The value of Tx_Err_Counter is then reset to zero.
- If FlexCAN is in ‘bus-off’ state, then Tx_Err_Counter is cascaded together with another internal counter to count the 128th occurrences of 11 consecutive recessive bits on the bus. Hence, Tx_Err_Counter is reset to 0 and counts in a manner where the internal counter counts 11 such bits and then wraps around while incrementing the Tx_Err_Counter. When Tx_Err_Counter reaches the value of 128, the FLT_CONF field in the error and status register is updated to be ‘error active’ and both error counters are reset to zero. At any instance of dominant bit following a stream of less

than 11 consecutive recessive bits, the internal counter resets itself to zero without affecting the Tx_Err_Counter value.

- If during system start-up, only one node is operating, then its Tx_Err_Counter increases in each message it is trying to transmit, as a result of acknowledge errors (indicated by the ACK_ERR bit in the error and status register). After the transition to ‘error passive’ state, the Tx_Err_Counter does not increment anymore by acknowledge errors. Therefore, the device never goes to the ‘bus-off’ state.
- If the Rx_Err_Counter increases to a value greater than 127, it is not incremented further, even if more errors are detected while being a receiver. At the next successful message reception, the counter is set to a value between 119 and 127 to resume to ‘error active’ state.

15.3.3.8 Error and Status Register (ESR)

This register reflects various error conditions, some general status of the device, and it is the source of four interrupts to the CPU. The reported error conditions (bits 15–10) are those that occurred since the last time the CPU read this register. The CPU read action clears bits 15–10. Bits 9–4 are status bits.

Most bits in this register are read only, except TWRN_INT, RWRN_INT, BOFF_INT, and ERR_INT, which are interrupt flags that can be cleared by writing ‘1’ to them (writing ‘0’ has no effect). See [Section 15.4.10, “Interrupts,”](#) for more details.

Offset 0x020

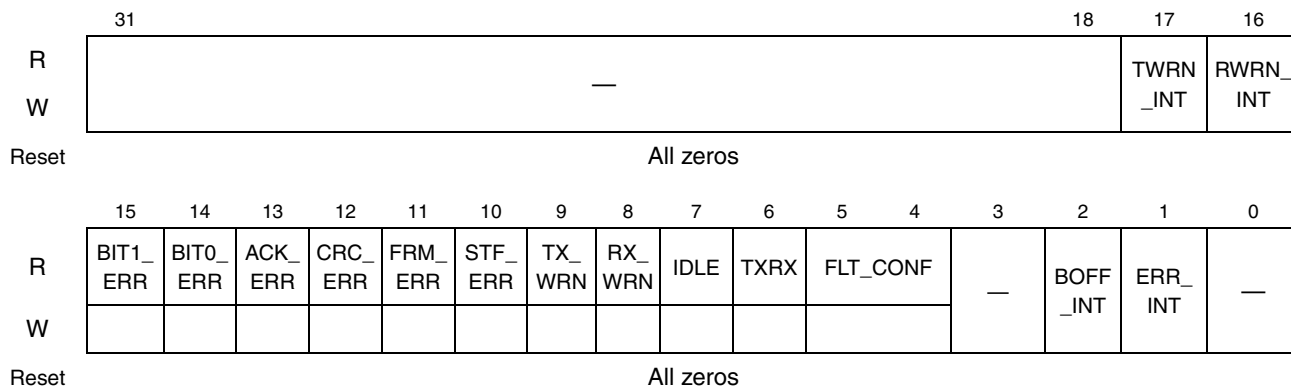


Figure 15-12. Error and Status Register (ESR)

Table 15-13 shows error and status register description.

Table 15-13. Error and Status Register Description

Bits	Name	Description
31–18	—	Reserved
17	TWRN_INT	Tx Warning Interrupt Flag: If the WRN_EN bit in MCR is asserted, the TWRN_INT bit is set when the TX_WRN flag transition from ‘0’ to ‘1’, meaning that the Tx error counter reached 96. If the corresponding mask bit in the control register (TWRN_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to ‘1’. Writing ‘0’ has no effect. 1 The Tx error counter transition from <96 to ≥96 0 No such occurrence

Table 15-13. Error and Status Register Description (continued)

Bits	Name	Description
16	RWRN_INT	Rx Warning Interrupt Flag: If the WRN_EN bit in MCR is asserted, the RWRN_INT bit is set when the RX_WRN flag transition from '0' to '1', meaning that the Rx error counters reached 96. If the corresponding mask bit in the Control Register (RWRN_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to '1'. Writing '0' has no effect. 1 The Rx error counter transition from < 96 to ≥ 96 0 No such occurrence
15	BIT1_ERR	Bit1 Error: This bit indicates when an inconsistency occurs between the transmitted and the received bit in a message. 1 At least one bit sent as recessive is received as dominant 0 No such occurrence Note: This bit is not set by a transmitter in case of arbitration field or ACK slot, or in case of a node sending a passive error flag that detects dominant bits.
14	BIT0_ERR	Bit0 Error: This bit indicates when an inconsistency occurs between the transmitted and the received bit in a message. 1 At least one bit sent as dominant is received as recessive 0 No such occurrence
13	ACK_ERR	Acknowledge Error: This bit indicates that an acknowledge error has been detected by the transmitter node, that is, a dominant bit has not been detected during the ACK SLOT. 1 An ACK error occurred since last read of this register 0 No such occurrence
12	CRC_ERR	Cyclic Redundancy Check Error: This bit indicates that a CRC Error has been detected by the receiver node, that is, the calculated CRC is different from the received. 1 A CRC error occurred since last read of this register. 0 No such occurrence
11	FRM_ERR	Form Error: This bit indicates that a form error has been detected by the receiver node, that is, a fixed-form bit field contains at least one illegal bit. 1 A form error occurred since last read of this register 0 No such occurrence
10	STF_ERR	Stuffing Error: This bit indicates that a Stuffing Error has been detected. 1 A Stuffing Error occurred since last read of this register. 0 No such occurrence.
9	TX_WRN	TX Error Warning: This bit indicates when repetitive errors are occurring during message transmission. 1 TX_Err_Counter ≥ 96 0 No such occurrence
8	RX_WRN	Rx Error Warning: This bit indicates when repetitive errors are occurring during message reception. 1 Rx_Err_Counter ≥ 96 0 No such occurrence
7	IDLE	CAN bus IDLE state: This bit indicates when CAN bus is in IDLE state. 1 CAN bus is now IDLE 0 No such occurrence
6	TXRX	Current FlexCAN status (transmitting/receiving): This bit indicates if FlexCAN is transmitting or receiving a message when the CAN bus is not in IDLE state. This bit has no meaning when IDLE is asserted. 1 FlexCAN is transmitting a message (IDLE=0) 0 FlexCAN is receiving a message (IDLE=0)

Table 15-13. Error and Status Register Description (continued)

Bits	Name	Description
5-4	FLT_CONF	Fault Confinement State: This 2-bit field indicates the confinement state of the FlexCAN module, as shown below. If the LOM bit in the control register is asserted, the FLT_CONF field indicates “Error Passive”. Since the control register is not affected by soft reset, the FLT_CONF field is not affected by soft reset if the LOM bit is asserted. 00 Error Active 01 Error Passive 1X Bus-off
3	—	Reserved
2	BOFF_INT	Bus-Off’ Interrupt: This bit is set when FlexCAN enters ‘bus-off’ state. If the corresponding mask bit in the control register (BOFF_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to ‘1’. Writing ‘0’ has no effect. 1 FlexCAN module entered ‘bus-off’ state 0 No such occurrence
1	ERR_INT	Error Interrupt: This bit indicates that at least one of the Error Bits (bits 15-10) is set. If the corresponding mask bit in the Control Register (ERR_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to ‘1’. Writing ‘0’ has no effect. 1 Indicates setting of any Error Bit in the Error and Status Register 0 No such occurrence
0	—	Reserved

15.3.3.9 Interrupt Masks 2 Register (IMASK2)

This register allows any number of a range of 32 message buffer interrupts to be enabled or disabled. It contains one interrupt mask bit per buffer, enabling the CPU to determine which buffer generates an interrupt after a successful transmission or reception (that is, when the corresponding IFLAG2 bit is set).

Figure 15-13 shows the interrupt masks 2 register (IMASK2).

Offset 0x024

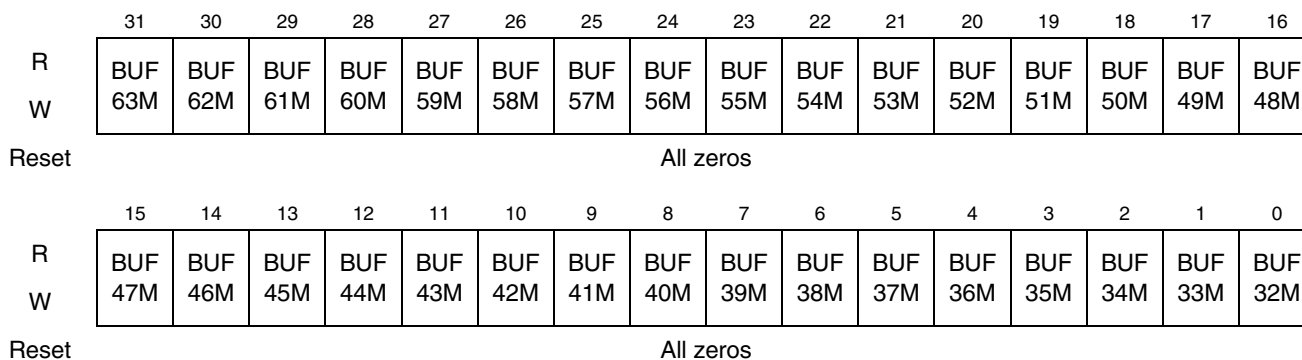


Figure 15-13. Interrupt Masks 2 Register (IMASK2)

Table 15-14 shows interrupt masks 2 register description.

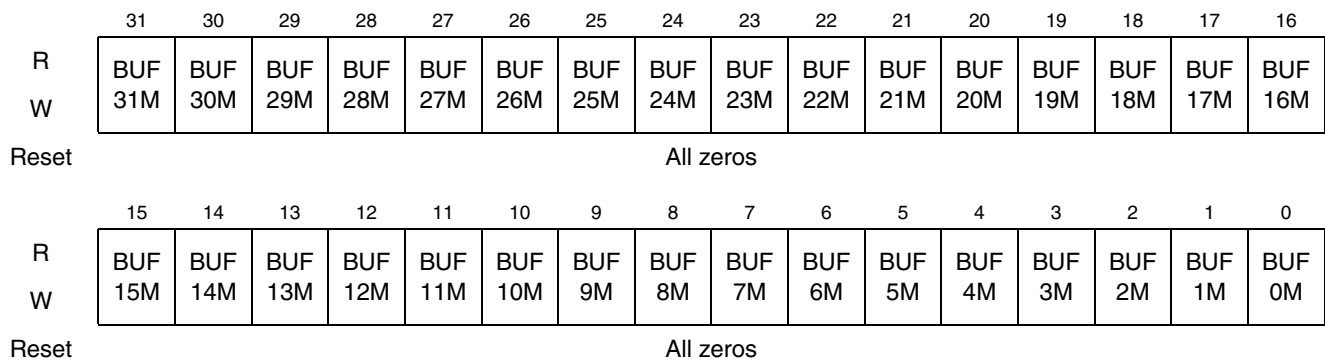
Table 15-14. Interrupt Masks 2 Register (IMASK2) Description

Bits	Name	Description
31–0	BUF63M–BUF32M	Buffer MB; Mask: Each bit enables or disables the respective FlexCAN Message Buffer (MB32 to MB63) Interrupt. 1 The corresponding buffer Interrupt is enabled 0 The corresponding buffer Interrupt is disabled Note: Setting or clearing a bit in the IMASK2 Register can assert or negate an interrupt request, if the corresponding IFLAG2 bit is set.

15.3.3.10 Interrupt Masks 1 Register (IMASK1)

This register allows to enable or disable any number of a range of 32 message buffer interrupts. It contains one interrupt mask bit per buffer, enabling the CPU to determine which buffer generates an interrupt after a successful transmission or reception (that is, when the corresponding IFLAG1 bit is set). [Figure 15-14](#) shows the interrupt masks 1 register (IMASK2).

Offset 0x028


Figure 15-14. Interrupt Masks 1 Register (IMASK1)

[Table 15-15](#) shows interrupt masks 2 register description.

Table 15-15. Interrupt Masks 1 Register (IMASK1) Description

Bits	Name	Description
31–0	BUF31M–BUF0M	Buffer MB; Mask: Each bit enables or disables the respective FlexCAN Message Buffer (MB0 to MB31) Interrupt. 1 The corresponding buffer Interrupt is enabled 0 The corresponding buffer Interrupt is disabled Note: Setting or clearing a bit in the IMASK1 Register can assert or negate an interrupt request, if the corresponding IFLAG1 bit is set.

15.3.3.11 Interrupt Flags 2 Register (IFLAG2)

This register defines the flags for 32 message buffer interrupts. It contains one interrupt flag bit per buffer. Each successful transmission or reception sets the corresponding IFLAG2 bit. If the corresponding IMASK2 bit is set, an interrupt is generated. The interrupt flag must be cleared by writing it to ‘1’. Writing ‘0’ has no effect.

When the AEN bit in the MCR is set (abort enabled), while the IFLAG2 bit is set for an MB configured as Tx, the writing access done by CPU into the corresponding MB is blocked.

Figure 15-15 shows the interrupt flags 2 register (IFLAG2).

Offset 0x02C

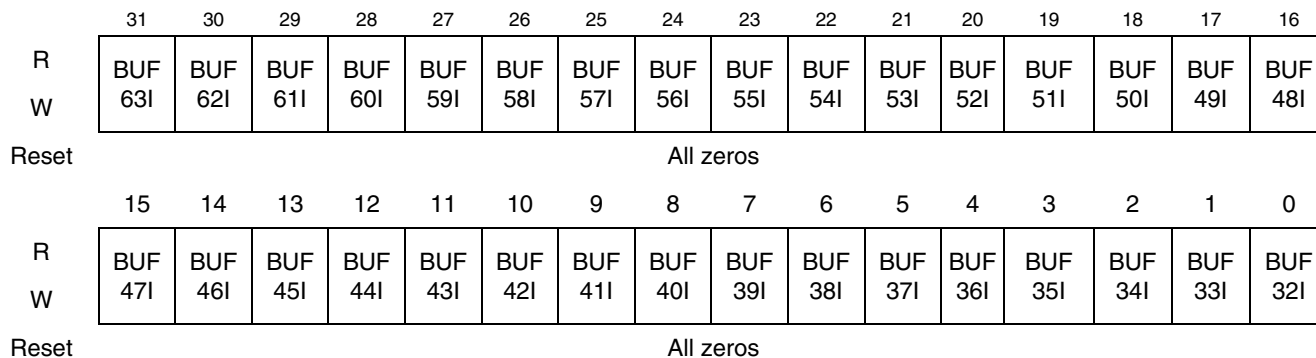


Figure 15-15. Interrupt Flags 2 Register (IFLAG2)

Table 15-16 shows interrupt flags 2 register (IFLAG2) description.

Table 15-16. Interrupt Flags 2 Register (IFLAG2) Description

Bits	Name	Description
31-0	BUF63I-BUF32I	Buffer MB; Mask: Each bit flags the respective FlexCAN Message Buffer (MB32 to MB63) interrupt. 1 The corresponding buffer has successfully completed transmission or reception 0 No such occurrence

15.3.3.12 Interrupt Flags 1 Register (IFLAG1)

This register defines the flags for 32 message buffer interrupts and FIFO interrupts. It contains one interrupt flag bit per buffer. Each successful transmission or reception sets the corresponding IFLAG1 bit. If the corresponding IMASK1 bit is set, an interrupt is generated. The interrupt flag must be cleared by writing it to '1'. Writing '0' has no effect.

When the AEN bit in the MCR is set (abort enabled), while the IFLAG1 bit is set for an MB configured as Tx, the writing access done by CPU into the corresponding MB is blocked.

When the FEN bit in the MCR is set (FIFO enabled), the function of the 8 least significant interrupt flags (BUF7I-BUF0I) is changed to support the FIFO operation. BUF7I, BUF6I, and BUF5I indicate operating conditions of the FIFO, while BUF4I to BUF0I are not used.

Figure 15-16 shows the interrupt flags 1 register (IFLAG1).

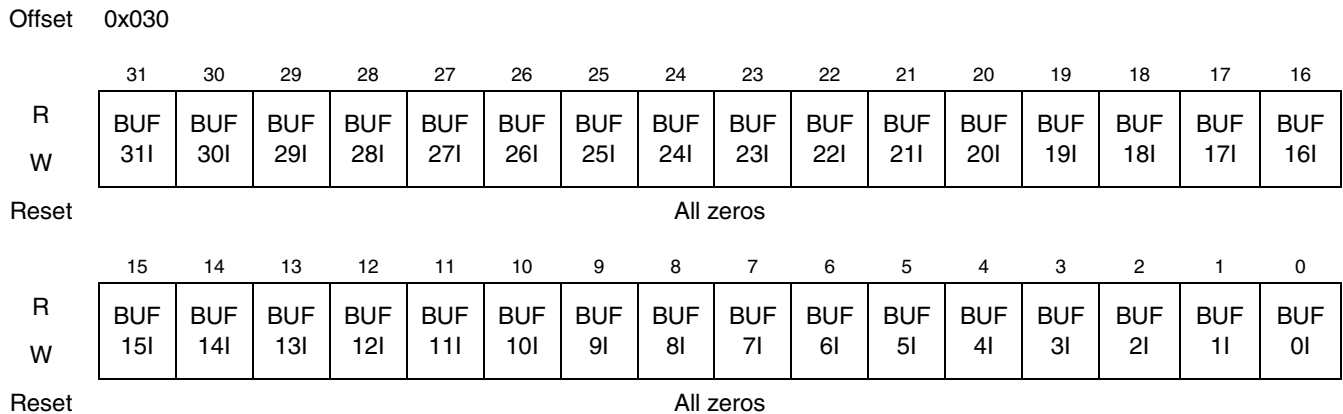


Figure 15-16. Interrupt Flags 1 Register (IFLAG1)

Table 15-17 shows interrupt flags 1 register (IFLAG1) description.

Table 15-17. Interrupt Flags 1 Register (IFLAG2) Description

Bits	Name	Description
31–8	BUF31I–BUF8I	Buffer MB _i Interrupt: Each bit flags the respective FlexCAN message buffer (MB8 to MB31) interrupt. 1 The corresponding MB has successfully completed transmission or reception 0 No such occurrence
7	BUF7I	Buffer MB7 Interrupt or “FIFO Overflow”: If the FIFO is not enabled, this bit flags the interrupt for MB7. If the FIFO is enabled, this flag indicates an overflow condition in the FIFO (frame lost because FIFO is full). 1 MB7 completed transmission/reception or FIFO overflow 0 No such occurrence
6	BUF6I	Buffer MB6 Interrupt or “FIFO Warning”: If the FIFO is not enabled, this bit flags the interrupt for MB6. If the FIFO is enabled, this flag indicates that 5 out of 6 buffers of the FIFO are already occupied (FIFO almost full). 1 MB6 completed transmission/reception or FIFO almost full 0 No such occurrence
5	BUF5I	Buffer MB5 Interrupt or “Frames available in FIFO”: If the FIFO is not enabled, this bit flags the interrupt for MB5. If the FIFO is enabled, this flag indicates that at least one frame is available to be read from the FIFO. 1 MB5 completed transmission/reception or frames available in the FIFO 0 No such occurrence
4–0	BUF4I–BUF0I	Buffer MB _i Interrupt or “reserved”: If the FIFO is not enabled, these bits flag the interrupts for MB0 to MB4. If the FIFO is enabled, these flags are not used and must be considered as reserved locations. 1 Corresponding MB completed transmission/reception 0 No such occurrence

15.3.3.13 Rx Individual Mask Registers (RXIMR0–RXIMR63)

These registers are used as acceptance masks for ID filtering in Rx MBs and the FIFO. If the FIFO is not enabled, one mask register is provided for each available message buffer, providing ID masking capability

on a per-message buffer basis. When the FIFO is enabled (FEN bit in MCR is set), the first 8 mask registers apply to the 8 elements of the FIFO filter table (on a one-to-one correspondence), while the rest of the registers apply to the regular MBs, starting from MB8.

The individual Rx mask registers are implemented in RAM, so they are not affected by reset and must be explicitly initialized prior to any reception. Furthermore, they can only be accessed by the CPU while the module is in freeze mode. Out of freeze mode, write accesses are blocked and read accesses returns “all zeros”. Furthermore, if the BCC bit in the MCR register is negated, any read or write operation to these registers results in access error.

NOTE

The individual Rx mask per message buffer feature may not be available in low-cost MCUs. Consult the specific MCU documentation to find out if this feature is supported. If not supported, the RXGMASK, RX14MASK, and RX15MASK registers are available, regardless of the value of the BCC bit.

Figure 15-17 shows the Rx individual mask registers (RXIMR0–RXIMR63).

Offset 0x880–0x97F

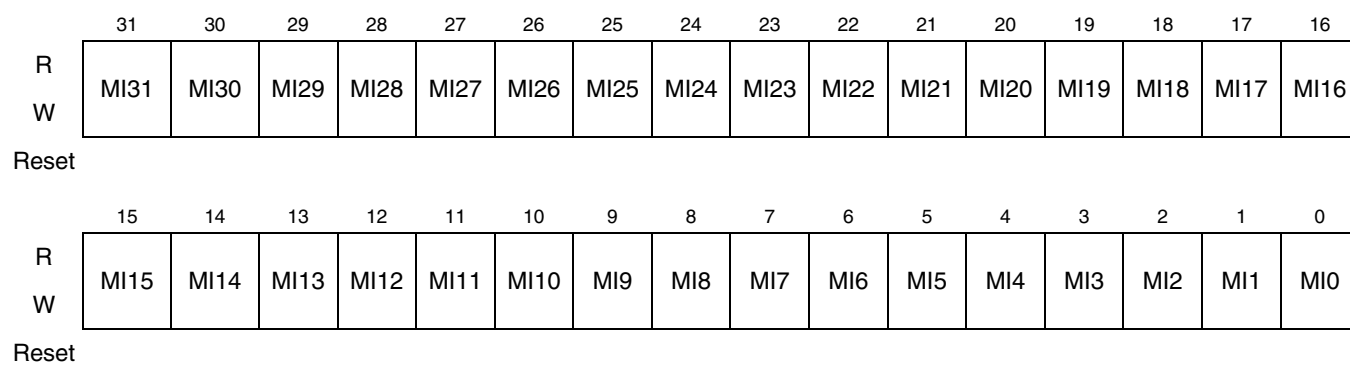


Figure 15-17. Rx Individual Mask Registers (RXIMR0–RXIMR63)

Table 15-16 shows interrupt flags 2 register (IFLAG2) description.

Table 15-18. Interrupt Flags 2 Register (IFLAG2) Description

Bits	Name	Description
31–0	MI31–MI0	Mask Bits: For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 1 The corresponding bit in the filter is checked against the one received 0 The corresponding bit in the filter is “don’t care”

15.4 Functional Description

15.4.1 Overview

The FlexCAN module is a CAN protocol engine with a very flexible mailbox system for transmitting and receiving CAN frames. The mailbox system is composed by a set of up to 64 MBs that store configuration

and control data, time stamp, message ID, and data (see [Section 15.3.1, “Message Buffer Structure”](#)). The memory corresponding to the first 8 MBs can be configured to support a FIFO reception scheme with a powerful ID filtering mechanism, capable of checking incoming frames against a table of IDs (up to 8 extended IDs or 16 standard IDs or 32 8-bit ID slices), each one with its own individual mask register. Simultaneous reception through FIFO and mailbox is supported. For mailbox reception, a matching algorithm makes it possible to store received frames only into MBs that have the same ID programmed on its ID field. A masking scheme makes it possible to match the ID programmed on the MB with a range of IDs on received CAN frames. For transmission, an arbitration algorithm decides the prioritization of MBs to be transmitted based on the message ID (optionally augmented by 3 local priority bits) or the MB ordering.

An MB is “active” at a given time if it can participate in the matching and arbitration algorithms that are occurring at that time. An Rx MB with a ‘0000’ code is inactive (refer to [Table 15-5](#)). Similarly, a Tx MB with a ‘1000’ or ‘1001’ code is also inactive (refer to [Table 15-6](#)). An MB not programmed with ‘0000’, ‘1000’, or ‘1001’ is temporarily deactivated (not participate in the current arbitration or matching run) when the CPU writes to the C/S field of that MB (see [Section 15.4.6.2, “Message Buffer Deactivation”](#)).

15.4.2 Transmit Process

To transmit a CAN frame, the CPU must prepare a message buffer for transmission by executing the following procedure:

1. If the MB is active (transmission pending), write an ABORT code (‘1001’) to the code field of the control and status word to request an abortion of the transmission, then read back the code field and the IFLAG register to check if the transmission was aborted (see [Section 15.4.6.1, “Transmission Abort Mechanism”](#)). If backwards compatibility is desired (AEN in MCR negated), just write ‘1000’ to the code field to inactivate the MB but then the pending frame is transmitted without notification (see [Section 15.4.6.2, “Message Buffer Deactivation”](#)).
2. Write the ID word.
3. Write the data bytes.
4. Write the length, control and code fields of the control and status word to activate the MB.

Once the MB is activated in the fourth step, it participates into the arbitration process and eventually be transmitted according to its priority. At the end of the successful transmission, the value of the free running timer is written into the time stamp field, the code field in the control and status word is updated, a status flag is set in the interrupt flag register and an interrupt is generated if allowed by the corresponding interrupt mask register bit. The new code field after transmission depends on the code that activates the MB in Step 4 (see [Table 15-5](#) and [Table 15-6](#) in [Section 15.3.1, “Message Buffer Structure”](#)). When the abort feature is enabled (AEN in MCR is asserted), after the interrupt flag is asserted for an MB configured as transmit buffer, the MB is blocked, therefore the CPU is not able to update it until the interrupt flag be negated by CPU. It means that the CPU must clear the corresponding IFLAG before starting to prepare this MB for a new transmission or reception.

15.4.3 Arbitration process

The arbitration process is an algorithm executed by the MBM that scans the whole MB memory looking for the highest priority message to be transmitted. All MBs programmed as transmit buffers are scanned to find the lowest ID¹ or the lowest MB number or the highest priority, depending on the LBUF and LPRIO_EN bits on the control register. The arbitration process is triggered during the following events:

- CRC field of the CAN frame
- Error delimiter field of the CAN frame
- Intermission, if the winner MB defined in a previous arbitration was deactivated, or if there was no MB to transmit, but the CPU wrote to the C/S word of any MB after the previous arbitration finished
- MBM is in idle or bus-off state and the CPU writes to the C/S word of any MB
- Upon leaving freeze mode

When LBUF is asserted, the LPRIO_EN bit has no effect and the lowest number buffer is transmitted first. When LBUF and LPRIO_EN are both negated, the MB with the lowest ID is transmitted first. However, if LBUF is negated and LPRIO_EN is asserted, the PRIO bits augment the ID used during the arbitration process. With this extended ID concept, arbitration is done based on the full 32-bit ID and the PRIO bits define that MB should be transmitted first, therefore MBs with PRIO=000 have higher priority. If two or more MBs have the same priority, the regular ID determines the priority of transmission. If two or more MBs have the same priority (3 extra bits) and the same regular ID, the lowest MB is transmitted first.

Once the highest priority MB is selected, it is transferred to a temporary storage space called serial message buffer (SMB) that has the same structure as a normal MB but is not user accessible. This operation is called “move-out”, and after it is done, write access to the corresponding MB is blocked (if the AEN bit in MCR is asserted).

The write access is released in the following events:

- After the MB is transmitted
- FlexCAN enters in HALT or bus-off
- FlexCAN loses the bus arbitration or there is an error during the transmission

At the first opportunity window on the CAN bus, the message on the SMB is transmitted according to the CAN protocol rules. FlexCAN transmits up to 8 data bytes, even if the data length code (DLC) value is bigger.

15.4.4 Receive Process

To be able to receive CAN frames into the mailbox MBs, the CPU must prepare one or more MBs for reception by executing the following steps:

- If the MB has a pending transmission, write an ABORT code (‘1001’) to the code field of the control and status word to request an abortion of the transmission, then read back the code field and the IFLAG register to check if the transmission was aborted (see [Section 15.4.6.1](#),

1. Actually, if LBUF is negated, the arbitration considers not only the ID, but also the RTR and IDE bits placed inside the ID at the same positions they are transmitted in the CAN frame.

“[Transmission Abort Mechanism](#)”). If backwards compatibility is desired (AEN in MCR negated), just write ‘1000’ to the code field to inactivate the MB, but then the pending frame may be transmitted without notification (see [Section 15.4.6.2, “Message Buffer Deactivation](#)”). If the MB already programmed as a receiver, just write ‘0000’ to the code field of the control and status word to keep the MB inactive.

- Write the ID word
- Write ‘0100’ to the code field of the control and status word to activate the MB

Once the MB is activated in the third step, it is able to receive frames that match the programmed ID. At the end of a successful reception, the MB is updated by the MBM as follows:

- The value of the free running timer is written into the time stamp field
- The received ID, data (8 bytes at most) and length fields are stored
- The code field in the control and status word is updated (see [Table 15-5](#) and [Table 15-6](#) in [Section 15.3.1, “Message Buffer Structure”](#))
- A status flag is set in the interrupt flag register and an interrupt is generated if allowed by the corresponding interrupt mask register bit

On receiving the MB interrupt, the CPU should service the received frame using the following procedure:

- Read the control and status word (mandatory—activates an internal clock for this buffer)
- Read the ID field (optional—needed only if a mask is used)
- Read the data field
- Read the free running timer (optional—releases the internal lock)

On reading the control and status word, if the BUSY bit is set in the code field, then the CPU should defer the access to the MB until this bit is negated. Reading the free running timer is not mandatory. If not executed the MB remains locked, unless the CPU reads the C/S word of another MB. Note that only a single MB is locked at a time. The only mandatory CPU read operation is the one on the control and status word to assure data coherency (see [Section 15.4.6, “Data Coherence](#)”).

The CPU should synchronize to frame reception by the status flag bit for the specific MB in one of the IFLAG registers and not by the code field of that MB. Polling the code field does not work because once a frame is received and the CPU services the MB (by reading the C/S word followed by unlocking the MB), the code field does not return to EMPTY. It remains FULL, as explained in [Table 15-5](#). If the CPU tries to workaround this behavior by writing to the C/S word to force an EMPTY code after reading the MB, the MB is actually deactivated from any currently ongoing matching process. As a result, a newly received frame matching the ID of that MB may be lost.

NOTE

Read the IFLAG registers rather than polling by reading directly the C/S word of the MBs

Received ID field is always stored in the matching MB, thus, the contents of the ID field in an MB may change if the match is due to masking. FlexCAN does receive frames transmitted by itself if there exists an Rx matching MB, provided the SRX_DIS bit in the MCR is not asserted. If SRX_DIS is asserted, FlexCAN does not store frames transmitted by itself in any MB, even if it contains a matching MB, and no interrupt flag or interrupt signal is generated due to the frame reception.

To receive CAN frames through the FIFO, the CPU must enable and configure the FIFO during freeze mode (see [Section 15.4.7, “Rx FIFO”](#)). Upon receiving the frames available interrupt from FIFO, the CPU should service the received frame using the following procedure:

- Read the control and status word (optional—needed only if a mask was used for IDE and RTR bits)
- Read the ID field (optional—needed only if a mask was used)
- Read the data field
- Clear the frames available interrupt (mandatory—release the buffer and allow the CPU to read the next FIFO entry)

15.4.5 Matching Process

The matching process is an algorithm executed by the MBM that scans the MB memory looking for Rx MBs programmed with the same ID as the one received from the CAN bus. If the FIFO is enabled, the 8-entry ID table from FIFO is scanned first and then, if a match is not found within the FIFO table, the other MBs are scanned. In the event that the FIFO is full, the matching algorithm always look for a matching MB outside the FIFO region.

When the frame is received, it is temporarily stored in a hidden auxiliary MB called Serial Message Buffer (SMB). The matching process takes place during the CRC field of the received frame. If a matching ID is found in the FIFO table or in one of the regular MBs, the contents of the SMB is transferred to the FIFO or to the matched MB during the 6th bit of the end-of-frame field of the CAN protocol. This operation is called “move-in”. If any protocol error (CRC, ACK) is detected, than the move-in operation does not happen.

For the regular mailbox MBs, an MB is said to be “free to receive” a new frame if the following conditions are satisfied:

- The MB is not locked (see [Section 15.4.6.3, “Message Buffer Lock Mechanism”](#))
- The Code field is either EMPTY or else it is FULL or OVERRUN but the CPU has already serviced the MB (read the C/S word and then unlocked the MB)

If the first MB with a matching ID is not “free to receive” the new frame, then the matching algorithm keeps looking for another free MB until it finds one. If it cannot find one that is free, then it overwrites the last matching MB (unless it is locked) and set the code field to OVERRUN (refer to [Table 15-5](#) and [Table 15-6](#)). If the last matching MB is locked, then the new message remains in the SMB, waiting for the MB to be unlocked (see [Section 15.4.6.3, “Message Buffer Lock Mechanism”](#)).

For example, FIFO is disabled and there are two MBs with the same ID, and FlexCAN starts receiving messages with that ID. Let us say that these MBs are the second and the fifth in the array. When the first message arrives, the matching algorithm finds the first match in MB number 2. The code of this MB is EMPTY, so the message is stored there. When the second message arrives, the matching algorithm finds MB number 2 again, but it is not “free to receive”, so it keeps looking and finding MB number 5 and store the message there. If yet another message with the same ID arrives, the matching algorithm finds out that there are no matching MBs that are “free to receive”, so it decides to overwrite the last matched MB, that is, number 5. In doing so, it sets the Code field of the MB to indicate OVERRUN.

The ability to match the same ID in more than one MB can be exploited to implement a reception queue (in addition to the full-featured FIFO) to allow more time for the CPU to service the MBs. By programming more than one MB with the same ID, received messages are queued into the MBs. The CPU can examine the Time Stamp field of the MBs to determine the order in which the messages arrived.

The matching algorithm described above can be changed to be the same one used in previous versions of the FlexCAN module. When the BCC bit in MCR is negated, the matching algorithm stops at the first MB with a matching ID that it finds, whether this MB is free or not. As a result, the message queuing feature does not work if the BCC bit is negated.

Matching to a range of IDs is possible by using ID acceptance Masks. FlexCAN supports individual masking per MB, refer to [Section 15.3.3.13, “Rx Individual Mask Registers \(RXIMR0–RXIMR63\).”](#) During the matching algorithm, if a mask bit is asserted, then the corresponding ID bit is compared. If the mask bit is negated, the corresponding ID bit is “don’t care”. Note that the individual mask registers are implemented in RAM, so they are not initialized out of reset. Also, they can only be programmed if the BCC bit is asserted and while the module is in freeze mode.

FlexCAN also supports an alternate masking scheme with only three mask registers (RGXMASK, RX14MASK, and RX15MASK) for backwards compatibility. This alternate masking scheme is enabled when the BCC bit in the MCR register is negated.

15.4.6 Data Coherence

To maintain data coherency and FlexCAN proper operation, the CPU must follow the rules described in [Section 15.4.2, “Transmit Process](#) and [Section 15.4.4, “Receive Process](#). Any form of CPU accessing an MB structure within FlexCAN other than those specified may cause FlexCAN to behave in an unpredictable way.

15.4.6.1 Transmission Abort Mechanism

The abort mechanism provides a safe way to request the abortion of a pending transmission. A feedback mechanism is provided to inform the CPU if the transmission was aborted or if the frame could not be aborted and was transmitted instead. To maintain backwards compatibility, the abort mechanism must be explicitly enabled by asserting the AEN bit in the MCR.

To abort a transmission, the CPU must write a specific abort code (1001) to the code field of the control and status word. When the abort mechanism is enabled, the active MBs configured as transmission must be aborted first and then they may be updated. If the abort code is written to an MB that is currently being transmitted, or to an MB that is already loaded into the SMB for transmission, the write operation is blocked and the MB is not deactivated, but the abort request is captured and kept pending until one of the following conditions is satisfied:

- The module loses the bus arbitration
- There is an error during the transmission
- The module is in freeze mode

If none of the above condition is attained, the MB is transmitted correctly, the interrupt flag is set in the IFLAG register, and an interrupt to the CPU is generated (if enabled). The abort request is automatically

cleared when the interrupt flag is set. On the other hand, if one of the above conditions is reached, the frame is not transmitted, therefore, the abort code is written into the code field, the interrupt flag is set in the IFLAG, and an interrupt is (optionally) generated to the CPU.

If the CPU writes the abort code before the transmission begins internally, then the write operation is not blocked, therefore, the MB is updated and no interrupt flag is set. This way the CPU just needs to read the abort code to make sure the active MB is deactivated. Although the AEN bit is asserted and the CPU wrote the abort code, in this case the MB is deactivated and not aborted, because the transmission did not start yet. One MB is only aborted when the abort request is captured and kept pending until one of the previous conditions are satisfied.

The abort procedure can be summarized as follows:

- CPU writes 1001 into the code field of the C/S word
- CPU reads the CODE field and compares it to the value that was written
- If the CODE field that was read is different from the value that was written, the CPU must read the corresponding IFLAG to check if the frame was transmitted or it is being currently transmitted. If the corresponding IFLAG is set, the frame was transmitted. If the corresponding IFLAG is reset, the CPU must wait for it to be set, and then the CPU must read the CODE field to check if the MB was aborted (CODE=1001) or it was transmitted (CODE=1000).

15.4.6.2 Message Buffer Deactivation

Deactivation is a mechanism provided to maintain data coherence when the CPU writes to the control and status word of active MBs out of freeze mode. Any CPU write access to the control and status word of an MB causes that MB to be excluded from the transmit or receive processes during the current matching or arbitration round. The deactivation is temporary, affecting only for the current match/arbitration round.

The purpose of deactivation is data coherency. The match/arbitration process scans the MBs to decide which MB to transmit or receive. If the CPU updates the MB in the middle of a match or arbitration process, the data of that MB may no longer be coherent, therefore deactivation of that MB is done.

Even with the coherence mechanism described above, writing to the control and status word of active MBs when not in freeze mode may produce undesirable results. Examples are:

- Matching and arbitration are one-pass processes. If MBs are deactivated after they are scanned, no re-evaluation is done to determine a new match/winner. If an Rx MB with a matching ID is deactivated during the matching process after it was scanned, then this MB is marked as invalid to receive the frame, and FlexCAN keeps looking for another matching MB within the ones it has not scanned yet. If it cannot find one, then the message is lost. Suppose, for example, that two MBs have a matching ID to a received frame, and the user deactivated the first matching MB after FlexCAN has scanned the second. The received frame is lost even if the second matching MB is “free to receive”.
- If a Tx MB containing the lowest ID is deactivated after FlexCAN has scanned it, then FlexCAN looks for another winner within the MBs that it has not scanned yet. Therefore, it may transmit an MB with ID that may not be the lowest at the time because a lower ID might be present in one of the MBs that it had already scanned before the deactivation.

- There is a point in time until which the deactivation of a Tx MB causes it not to be transmitted (end of move-out). After this point, it is transmitted but no interrupt is issued and the Code field is not updated. To avoid this situation, the abort procedures described in [Section 15.4.6.1, “Transmission Abort Mechanism”](#) should be used.

15.4.6.3 Message Buffer Lock Mechanism

Besides MB deactivation, FlexCAN has another data coherence mechanism for the receive process. When the CPU reads the control and status word of an “active not empty” Rx MB, FlexCAN assumes that the CPU wants to read the whole MB in an atomic operation, and thus it sets an internal lock flag for that MB. The lock is released when the CPU reads the Free Running Timer (global unlock operation), or when it reads the control and status word of another MB. The MB locking is done to prevent a new frame to be written into the MB while the CPU is reading it.

NOTE

The locking mechanism only applies to Rx MBs that have a code different than INACTIVE (‘0000’) or EMPTY¹ (‘0100’). Also, Tx MBs cannot be locked.

Suppose, for example, that the FIFO is disabled and the second and the fifth MBs of the array are programmed with the same ID, and FlexCAN has already received and stored messages into these two MBs. Suppose now that the CPU decides to read MB number 5 and at the same time another message with the same ID is arriving. When the CPU reads the control and status word of MB number 5, this MB is locked. The new message arrives and the matching algorithm finds out that there are no “free to receive” MBs, so it decides to override MB number 5. However, this MB is locked, so the new message cannot be written there. It remains in the SMB waiting for the MB to be unlocked, and only then it is written to the MB. If the MB is not unlocked in time and yet another new message with the same ID arrives, then the new message overwrites the one on the SMB and there is no indication of lost messages either in the code field of the MB or in the error and status register.

While the message is being moved-in from the SMB to the MB, the BUSY bit on the Code field is asserted. If the CPU reads the control and status word and finds out that the BUSY bit is set, it should defer accessing the MB until the BUSY bit is negated.

NOTE

If the BUSY bit is asserted or if the MB is empty, then reading the control and status word does not lock the MB.

Deactivation takes precedence over locking. If the CPU deactivates a locked Rx MB, then its lock status is negated and the MB is marked as invalid for the current matching round. Any pending message on the SMB is not transferred anymore to the MB.

1. In previous FlexCAN versions, reading the C/S word locked the MB even if it was EMPTY. This behavior is honored when the BCC bit is negated.

15.4.7 Rx FIFO

The receive-only FIFO is enabled by asserting the FEN bit in the MCR. The reset value of this bit is zero to maintain software backwards compatibility with previous versions of the module that did not have the FIFO feature. When the FIFO is enabled, the memory region normally occupied by the first 8 MBs (0x80–0xFF) is now reserved for use of the FIFO engine (see [Section 15.3.2, “Rx FIFO Structure”](#)). Management of read and write pointers is done internally by the FIFO engine. The CPU can read the received frames sequentially, in the order they were received, by repeatedly accessing a Message Buffer structure at the beginning of the memory.

The FIFO can store up to 6 frames pending service by the CPU. An interrupt is sent to the CPU when new frames are available in the FIFO. Upon receiving the interrupt, the CPU must read the frame (accessing an MB in the 0x80 address) and then clear the interrupt. The act of clearing the interrupt triggers the FIFO engine to replace the MB in 0x80 with the next frame in the queue, and then issue another interrupt to the CPU. If the FIFO is full and more frames continue to be received, an OVERFLOW interrupt is issued to the CPU and subsequent frames are not accepted until the CPU creates space in the FIFO by reading one or more frames. A warning interrupt is also generated when 5 frames are accumulated in the FIFO.

A powerful filtering scheme is provided to accept only frames intended for the target application, thus reducing the interrupt servicing work load. The filtering criteria is specified by programming a table of eight 32-bit registers that can be configured to one of the following formats (see also [Section 15.3.2, “Rx FIFO Structure”](#)):

- Format A: 8 extended or standard IDs (including IDE and RTR)
- Format B: 16 standard IDs or 16 extended 14-bit ID slices (including IDE and RTR)
- Format C: 32 standard or extended 8-bit ID slices

NOTE

A chosen format is applied to all 8 registers of the filter table. It is not possible to mix formats within the table.

The eight elements of the filter table are individually affected by the first eight Individual Mask Registers (RXIMR0–RXIMR7), allowing very powerful filtering criteria to be defined. The rest of the RXIMR, starting from RXIM8, continue to affect the regular MBs, starting from MB8. If the BCC bit is negated (or if the RXIMR are not available for the particular MCU), then the FIFO filter table is affected by the legacy mask registers as follows: element 6 is affected by RX14MASK, element 7 by RX15MASK, and the other elements (0–5) by RXGMASK.

15.4.8 CAN Protocol Related Features

15.4.8.1 Remote Frames

Remote frame is a special kind of frame. The user can program an MB to be a request remote frame by writing the MB as transmit with the RTR bit set to 1. After the remote request frame is transmitted successfully, the MB becomes a receive message buffer, with the same ID as before.

When a remote request frame is received by FlexCAN, its ID is compared to the IDs of the transmit message buffers with the code field '1010'. If there is a matching ID, then this MB frame is transmitted. Note that if the matching MB has the RTR bit set, then FlexCAN transmits a remote frame as a response.

A received remote request frame is not stored in a receive buffer. It is only used to trigger a transmission of a frame in response. The mask registers are not used in remote frame matching, and all ID bits (except RTR) of the incoming received frame should match.

In the case that a remote request frame is received and matched an MB, this message buffer immediately enters the internal arbitration process, but is considered as normal Tx MB, with no higher priority. The data length of this frame is independent of the DLC field in the remote frame that initiated its transmission.

If the Rx FIFO is enabled (bit FEN set in MCR), FlexCAN does not generate an automatic response for remote request frames that match the FIFO filtering criteria. If the remote frame matches one of the target IDs, it is stored in the FIFO and presented to the CPU. Note that for filtering formats A and B, it is possible to select whether remote frames are accepted or not. For format C, remote frames are always accepted (if they match the ID).

15.4.8.2 Overload Frames

FlexCAN does transmit overload frames due to detection of following conditions on CAN bus:

- Detection of a dominant bit in the first/second bit of intermission
- Detection of a dominant bit at the 7th bit (last) of end of frame field (Rx frames)
- Detection of a dominant bit at the 8th bit (last) of error frame delimiter or overload frame delimiter

15.4.8.3 Time Stamp

The value of the free running timer is sampled at the beginning of the identifier field on the CAN bus, and is stored at the end of “move-in” in the TIME STAMP field, providing network behavior with respect to time.

Note that the free running timer can be reset upon a specific frame reception, enabling network time synchronization. Refer to TSYN description in [Section 15.3.3.2, “Control Register \(CTRL\).”](#)

15.4.8.4 Protocol Timing

Figure 15-18 shows the structure of the clock generation circuitry.

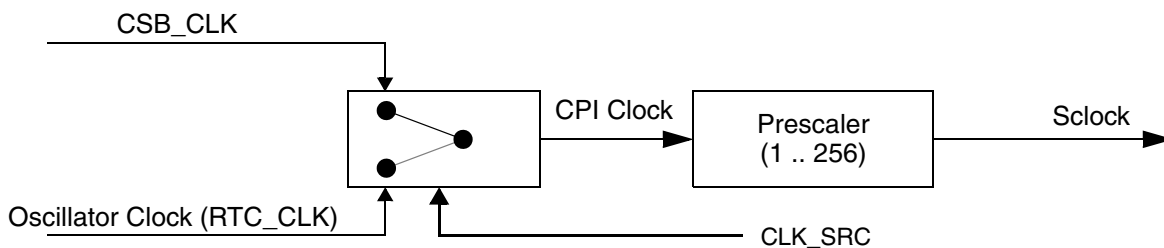


Figure 15-18. CAN Engine Clocking Scheme

The FlexCAN module supports a variety of means to set up bit timing parameters that are required by the CAN protocol. The control register has various fields used to control bit timing parameters: PRESDIV, PROPSEG, PSEG1, PSEG2, and RJW. See [Section 15.3.3.2, “Control Register \(CTRL\).”](#)

The PRESDIV field controls a prescaler that generates the Slock, whose period defines the ‘time quantum’ used to compose the CAN waveform. A time quantum is the atomic unit of time handled by the CAN engine.

$$f_{Tq} = \frac{f_{CANCLK}}{\text{(Prescaler value)}}$$

A bit time is subdivided into three segments¹ (reference [Figure 15-19](#) and [Table 15-19](#)):

- SYNC_SEG: This segment has a fixed length of one time quantum. Signal edges are expected to occur within this section
- Time Segment 1: This segment includes the propagation segment and the phase segment 1 of the CAN standard. It can be programmed by setting the PROPSEG and the PSEG1 fields of the CTRL register so that their sum (plus 2) is in the range of 4 to 16 time quanta

Time Segment 2: This segment represents the phase segment 2 of the CAN standard. It can be programmed by setting the PSEG2 field of the CTRL register (plus 1) to be 2–8 time quanta long

$$\text{Bit Rate} = \frac{f_{Tq}}{\text{(number of Time Quanta)}}$$

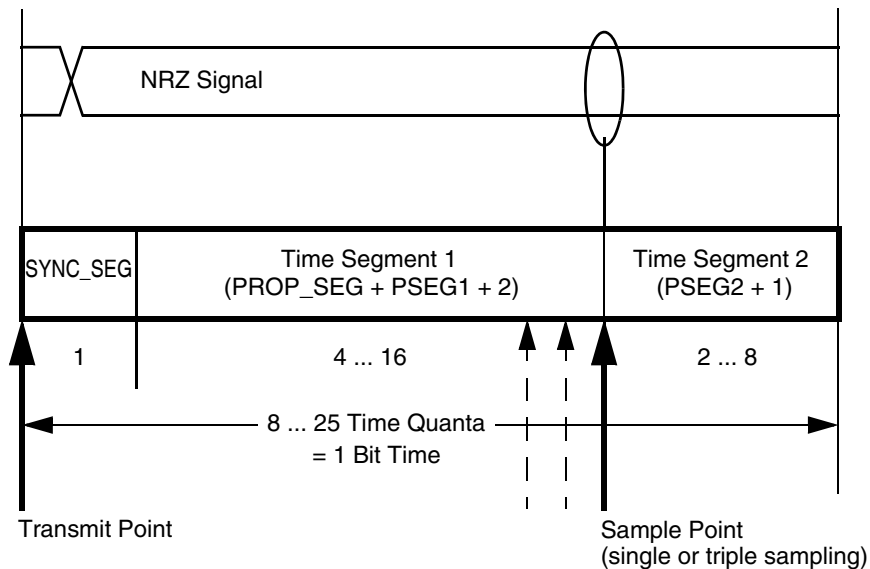


Figure 15-19. Segments within the Bit Time

1. For further explanation of the underlying concepts, refer to ISO/DIS 11519–1, Section 10.3. Reference also the Bosch CAN 2.0A/B protocol specification dated September 1991 for bit timing.

Table 15-19. Time Segment Syntax

Syntax	Description
SYNC_SEG	System expects transitions to occur on the bus during this period.
Transmit Point	A node in transmit mode transfers a new value to the CAN bus at this point.
Sample Point	A node samples the bus at this point. If the three samples per bit option is selected, then this point marks the position of the third sample.

Table 15-20 gives an overview of the CAN compliant segment settings and the related parameter values.

Table 15-20. CAN Standard Compliant Bit Time Segment Settings

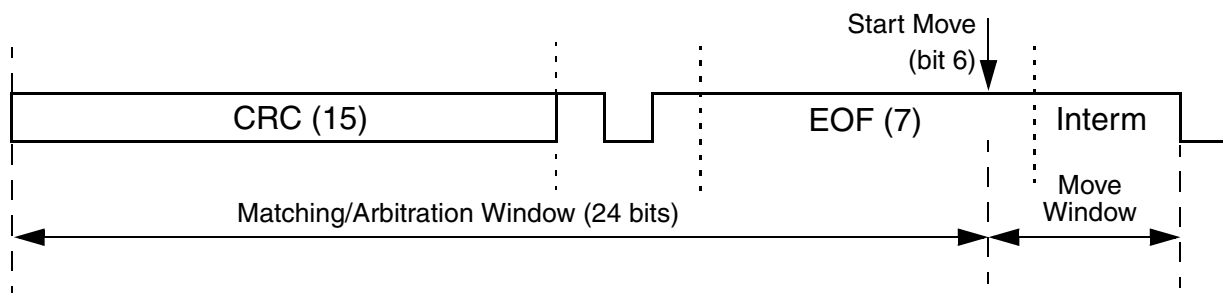
Time Segment 1	Time Segment 2	Re-synchronization Jump Width
5..10	2	1..2
4..11	3	1..3
5..12	4	1..4
6..13	5	1..4
7..14	6	1..4
8..15	7	1..4
9..16	8	1..4

NOTE

It is the user's responsibility to ensure the bit time settings are in compliance with the CAN standard. For bit time calculations, use an information processing time (IPT) of 2, that is the value implemented in the FlexCAN module.

15.4.8.5 Arbitration and Matching Timing

During normal transmission or reception of frames, the arbitration, matching, move-in and move-out processes are executed during certain time windows inside the CAN frame, as shown in Figure 15-20.


Figure 15-20. Arbitration, Match, and Move Time Windows

When doing matching and arbitration, FlexCAN needs to scan the whole message buffer memory during the available time slot. To have sufficient time to do that, the following requirements must be observed:

- A valid CAN bit timing must be programmed, as indicated in Table 15-20

- The peripheral clock frequency cannot be smaller than the oscillator clock frequency, that is, the PLL cannot be programmed to divide down the oscillator clock
- There must be a minimum ratio between the peripheral clock frequency and the CAN bit rate, as specified in [Table 15-21](#)

Table 15-21. Minimum Ratio Between Peripheral Clock Frequency and CAN Bit Rate

Number of Message Buffers	Minimum Ratio
16	8
32	8
64	16

A direct consequence of the first requirement is that the minimum number of time quanta per CAN bit must be 8, so the oscillator clock frequency should be at least 8 times the CAN bit rate. The minimum frequency ratio specified in [Table 15-21](#) can be achieved by choosing a high enough peripheral clock frequency when compared to the oscillator clock frequency, or by adjusting one or more of the bit timing parameters (PRES DIV, PROPSEG, PSEG1, and PSEG2). As an example, taking the case of 64 MBs, if the oscillator and peripheral clock frequencies are equal and the CAN bit timing is programmed to have 8 time quanta per bit, then the prescaler factor (PRES DIV + 1) should be at least 2. For prescaler factor equal to one and CAN bit timing with 8 time quanta per bit, the ratio between peripheral and oscillator clock frequencies should be at least 2.

15.4.9 Modes of Operation Details

15.4.9.1 Freeze Mode

This mode is entered by asserting the HALT bit in the MCR register or when the MCU is put into Debug mode. In both cases it is also necessary that the FRZ bit is asserted in the MCR register and the module is not in any of the low-power modes (disable, doze, and stop). When freeze mode is requested during transmission or reception, FlexCAN does the following:

- Waits to be in either intermission, passive error, bus-off or idle state
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish
- Ignores the Rx input pin and drives the Tx pin as recessive
- Stops the prescaler, thus halting all CAN protocol activities
- Grants write access to the error counters register, that is read-only in other modes
- Sets the NOT_RDY and FRZ_ACK bits in MCR

After requesting freeze mode, the user must wait for the FRZ_ACK bit to be asserted in MCR before executing any other action, otherwise FlexCAN may operate in an unpredictable way. In freeze mode, all memory-mapped registers are accessible.

Exiting freeze mode is done in one of the following ways:

- CPU negates the FRZ bit in the MCR Register

- The MCU is removed from Debug Mode and/or the HALT bit is negated

Once out of freeze mode, FlexCAN tries to re-synchronize to the CAN bus by waiting for 11 consecutive recessive bits.

15.4.9.2 Module Disable Mode

This low-power mode is entered when the MDIS bit in the MCR register is asserted. If the module is disabled during freeze mode, it shuts down the clocks to the CPI and MBM sub-modules, sets the LPM_ACK bit, and negates the FRZ_ACK bit. If the module is disabled during transmission or reception, FlexCAN does the following:

- Waits to be in either idle or bus-off state, or else waits for the third bit of intermission and then checks it to be recessive
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish
- Ignores its Rx input pin and drives its Tx pin as recessive
- Shuts down the clocks to the CPI and MBM sub-modules
- Sets the NOT_RDY and LPM_ACK bits in MCR

The BIU continues to operate, enabling the CPU to access memory-mapped registers, except the free running timer, the ECR and the message buffers, that cannot be accessed when the module is in disable mode. Exiting from this mode is done by negating the MDIS bit that resumes the clocks and negate the LPM_ACK bit.

15.4.10 Interrupts

The module can generate up to 69 interrupt sources (64 interrupts due to message buffers and 5 interrupts due to zeroed interrupts from MBs, bus-off, error, Tx warning, Rx warning). The number of actual sources depends on the configured number of message buffers.

Each one of the message buffers can be an interrupt source, if its corresponding IMASK bit is set. There is no distinction between Tx and Rx interrupts for a particular buffer, under the assumption that the buffer is initialized for either transmission or reception. Each of the buffers has assigned a flag bit in the IFLAG registers. The bit is set when the corresponding buffer completes a successful transmission/reception and is cleared when the CPU writes it to '1' (unless another interrupt is generated at the same time).

NOTE

It must be guaranteed that the CPU only clears the bit causing the current interrupt. For this reason, bit manipulation instructions (BSET) must not be used to clear interrupt flags. These instructions may cause accidental clearing of interrupt flags that are set after entering the current interrupt service routine.

If the Rx FIFO is enabled (bit FEN on MCR set), the interrupts corresponding to MBs 0–7 have a different behavior. Bit 7 of the IFLAG1 becomes the “FIFO overflow” flag; bit 6 becomes the FIFO warning flag, bit 5 becomes the “frames available in FIFO flag” and bits 4–0 are unused. See [Section 15.3.3.12, “Interrupt Flags 1 Register \(IFLAG1\),”](#) for more information.

A combined interrupt for all MBs is also generated by an Or of all the interrupt sources from MBs. This interrupt gets generated when any of the MBs generates an interrupt. In this case the CPU must read the IFLAG registers to determine which MB caused the interrupt.

The other 5 interrupt sources (bus-off, Error, Tx Warning, Rx Warning, and Wake-Up) generate interrupts like the MB ones, and can be read from the error and status register. The bus-off, Error, Tx Warning, and Rx Warning interrupt mask bits are located in the control register, and the wake-up interrupt mask bit is located in the MCR.

15.4.11 Bus Interface

The FlexCAN module interrupts are coalesced into a single interrupt and is available in the interrupt controller. The CAN_INT_STAT register maintains the status of Ored interrupts from MBs, bus-off, error, Tx warning, Rx warning for all the four FlexCAN modules in a single register. The CPU access to FlexCAN registers are subject to the following rules:

- Read and write access to unimplemented or reserved address space also results in access error. Any access to unimplemented MB or Rx individual mask register locations results in access error. Any access to the Rx individual mask register space when the BCC bit in MCR is negated results in access error.
- If MAXMB is programmed with a value smaller than the available number of MBs, then the unused memory space can be used as general purpose RAM space. Note that the Rx Individual mask registers can only be accessed in freeze mode, and this is still true for unused space within this memory. Note also that reserved words within RAM cannot be used. As an example, suppose FlexCAN is configured with 64 MBs and MAXMB is programmed with 0. The maximum number of MBs in this case becomes 1. The MB memory starts at 0x0060, but the space from 0x0060 to 0x007F is reserved (for SMB usage), and the space from 0x0080 to 0x008F is used by the one MB. This leaves us with the available space from 0x0090 to 0x047F. The available memory in the mask registers space would be from 0x0884 to 0x097F.

NOTE

Unused MB space must not be used as general purpose RAM while FlexCAN is transmitting and receiving CAN frames.

15.5 Initialization/Application Information

This section provide instructions for initializing the FlexCAN module.

15.5.1 FlexCAN Initialization Sequence

The FlexCAN module can be reset in following three ways:

- MCU level hard reset, which resets all memory-mapped registers asynchronously
- MCU level soft reset, which resets some of the memory-mapped registers synchronously (refer to [Table 15-2](#) to see what registers are affected by soft reset)
- SOFT_RST bit in MCR, which has the same effect as the MCU level soft reset

Soft reset is synchronous and has to follow an internal request/acknowledge procedure across clock domains. Therefore, it may take some time to fully propagate its effects. The SOFT_RST bit remains asserted while soft reset is pending, so software can poll this bit to know when the reset has completed. Also, soft reset cannot be applied while clocks are shut down in any of the low-power modes. The low-power mode should be exited and the clocks resumed before applying soft reset.

The clock source (CLK_SRC bit) should be selected while the module is in disable mode. After the clock source is selected and the module is enabled (MDIS bit negated), FlexCAN automatically goes to freeze mode. In freeze mode, FlexCAN is un-synchronized to the CAN bus, the HALT and FRZ bits in MCR register are set, the internal state machines are disabled and the FRZ_ACK and NOT_RDY bits in the MCR register are set. The Tx pin is in recessive state and FlexCAN does not initiate any transmission or reception of CAN frames. Note that the message buffers and the Rx individual mask registers are not affected by reset, so they are not automatically initialized.

For any configuration change/initialization, it is required that FlexCAN is put into freeze mode (see [Section 15.4.9.1, “Freeze Mode”](#)). The following is a generic initialization sequence applicable to the FlexCAN module:

- Initialize the module configuration register
 - Enable the individual filtering per MB and reception queue features by setting the BCC bit
 - Enable the warning interrupts by setting the WRN_EN bit
 - If required, disable frame self reception by setting the SRX_DIS bit
 - Enable the FIFO by setting the FEN bit
 - Enable the abort mechanism by setting the AEN bit
 - Enable the local priority feature by setting the LPRIO_EN bit
- Initialize the control register
 - Determine the bit timing parameters: PROPSEG, PSEG1, PSEG2, RJW
 - Determine the bit rate by programming the PRESDIV field
 - Determine the internal arbitration mode (LBUF bit)
- Initialize the message buffers
 - The control and status word of all message buffers must be initialized
 - If FIFO was enabled, the 8-entry ID table must be initialized
 - Other entries in each message buffer should be initialized as required
- Initialize the Rx individual mask registers
- Set required interrupt mask bits in the IMASK registers (for all MB interrupts), in CTRL register (for bus-off and error interrupts), and in MCR register for wake-up interrupt
- Negate the HALT bit in MCR

Starting with the last event, FlexCAN attempts to synchronize to the CAN bus.

15.5.2 FlexCAN Addressing and RAM size configurations

There are three RAM configurations that can be implemented within the FlexCAN module. The possible configurations are:

- For 16 MBs: 288 bytes for MB memory and 64 bytes for individual mask registers
- For 32 MBs: 544 bytes for MB memory and 128 bytes for individual mask registers
- For 64 MBs: 1056 bytes for MB memory and 256 bytes for individual mask registers

In each configuration the user can program the maximum number of MBs that takes part in the matching and arbitration processes using the MAXMB field in the MCR register.

- For 16 MB configuration, MAXMB can be any number between 0–15.
- For 32 MB configuration, MAXMB can be any number between 0–31.
- For 64 MB configuration, MAXMB can be any number between 0–63.

Chapter 16

Universal Serial Bus Interface

This chapter describes the universal serial bus (USB) interface of the device. The USB interface implements many industry standards. However, it is beyond the scope of this document to document the intricacies of these standards. Instead, it is left to the reader to refer to the governing specifications.

The following documents are available from the USB Implementers Forum web page at <http://www.usb.org/developers/docs/>.

- *Universal Serial Bus Revision 2.0 Specification*
- *On-the-go Supplement to the USB 2.0 Specification, Revision 1.0a*

The following documents are available from the Intel USB Specifications web page at <http://www.intel.com/technology/usb/spec.htm>.

- *Enhanced Host Controller Interface (EHCI) Specification for Universal Serial Bus, Revision 1.0*

The following documents are available from the ULPI web page at <http://www.ulpi.org/>.

- *UTMI + Specification, Revision 1.0*
- *UTMI Low Pin-Count Interface (ULPI) Specification, Revision 1.0*

16.1 Introduction

The device implements a dual-role (DR) USB module. This module may be connected to an external port. Collectively the module and external port are called the USB interface. The USB interface is shown in [Figure 16-1](#).

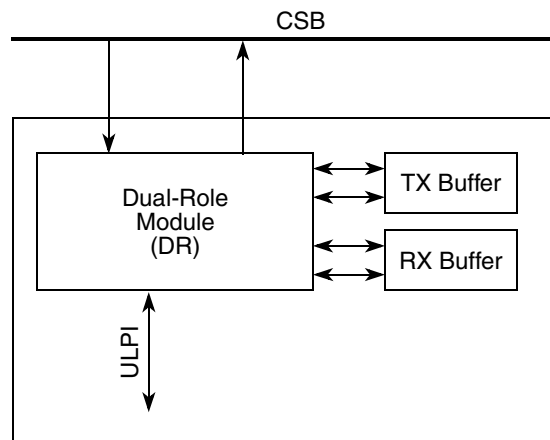


Figure 16-1. USB Interface Block Diagram

16.1.1 Overview

The USB DR module is a USB 2.0-compliant serial interface engine for implementing a USB interface. The registers and data structures for the module are based on the *Enhanced Host Controller Interface Specification for Universal Serial Bus* (EHCI) from Intel Corporation. The DR module can act as a device or host controller. Interfaces to negotiate the host or device role on the bus in compliance with the On-the-go (OTG) supplement to the USB specification are also provided.

The DR module supports the required signaling for UTMI low pin count interface (ULPI) transceivers (PHYs). An external PHY would be used to interface to ULPI.

The module contains a chaining DMA (direct memory access) engine that reduces the interrupt load on the application processor and reduces the total system bus bandwidth that must be dedicated to servicing the USB interface requirements.

16.1.2 Features

The USB DR module includes the following features:

- Complies with USB specification rev 2.0
- Supports operation as a standalone USB host controller
 - Supports enhanced host controller interface (EHCI)
- Supports high-speed (480 Mbps), full-speed (12 Mbps), and low-speed (1.5 Mbps) operation. Low speed is only supported in host mode.
- Supports external PHY with ULPI (UTMI + low-pin interface)
- Supports operation as a standalone USB device
 - Supports one upstream facing port
 - Supports three programmable, bidirectional USB endpoints
- Host and device support
- OTG (on-the-go) support, which includes both device and host capability, with external PHY (ULPI)

16.1.3 Modes of Operation

The USB DR module has three basic operating modes: host, device, and OTG.

NOTE

Only high-speed and full-speed operations are supported in device mode.

16.2 External Signals

This section contains detailed descriptions of all the USB dual-role controller signals.

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 bidirectional, generic data bus that carries USB and register data. This interface minimizes pin count requirements for external PHYs. [Table 16-1](#) describes the signals for the ULPI interface.

Table 16-1. ULPI Signal Descriptions

Signal	I/O	Description	
USBDR_DIR	I	Direction. USBDR_DIR controls the direction of the data bus. When the PHY has data to transfer to USB port, it drives USBDR_DIR high to take ownership of the bus. When the PHY has no data to transfer it drives USBDR_DIR low and monitors the bus for link activity. The PHY pulls USBDR_DIR high whenever the interface cannot accept data from the link.	
		State Meaning	Asserted—PHY has data to transfer to the link. Negated—PHY has no data to transfer.
		Timing	Synchronous to PHY_CLK.
USBDR_NXT	I	Next data. The PHY asserts USBDR_NXT to throttle the data. When USB port is sending data to the PHY, USBDR_NXT indicates when the current byte has been accepted by the PHY. The USB port places the next byte on the data bus in the following clock cycle. When the PHY is sending data to USB port, USBDR_NXT indicates when a new byte is available for USB port to consume.	
		State Meaning	Asserted—PHY is ready to transfer byte. Negated—PHY is not ready.
		Timing	Synchronous to PHY_CLK.
USBDR_STP	O	Stop. USBDR_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, USBDR_STP indicates the last byte of data was previously on the bus. If the PHY is sending data to USB port, USBDR_STP forces the PHY to end its transfer, negate USBDR_DIR and relinquish control of the data bus to the USB port. Negated—Indicates normal operation.
		Timing	Synchronous to PHY_CLK.
USBDR_PWR_FAULT	I	Power fault. USBDR_PWR_FAULT 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 PHY_CLK.
USBDR_PCTL0	O	Port control 0. USBDR_PCTL0 controls the port status indicator LED 0 when in host mode.	
		State Meaning	Asserted—LED on. Negated—LED off.
		Timing	Synchronous to PHY_CLK.

Table 16-1. ULPI Signal Descriptions (continued)

Signal	I/O	Description		
USBDR_PCTL1	O	Port control 1. USBDR_PCTL1 controls the port status indicator LED 1 when in host mode.		
		<table border="1"> <tr> <td>State Meaning</td> <td>Asserted—LED on. Negated—LED off.</td> </tr> </table>	State Meaning	Asserted—LED on. Negated—LED off.
		State Meaning	Asserted—LED on. Negated—LED off.	
<table border="1"> <tr> <td>Timing</td> <td>Synchronous to PHY_CLK.</td> </tr> </table>	Timing	Synchronous to PHY_CLK.		
Timing	Synchronous to PHY_CLK.			
USBDR_TXDRXD[0:7]	I/O	Data bit n . USBDR_TXDRXD n is bit n of the 8-bit (USBDR_TXDRXD7–USBDR_TXDRXD0), uni-directional data bus used to carry USB, register, and interrupt data between the PHY and the USB controller.		
		<table border="1"> <tr> <td>State Meaning</td> <td>Asserted—Data bit n is 1. Negated—Data bit n is 0.</td> </tr> </table>	State Meaning	Asserted—Data bit n is 1. Negated—Data bit n is 0.
		State Meaning	Asserted—Data bit n is 1. Negated—Data bit n is 0.	
<table border="1"> <tr> <td>Timing</td> <td>Synchronous to PHY_CLK.</td> </tr> </table>	Timing	Synchronous to PHY_CLK.		
Timing	Synchronous to PHY_CLK.			
USBDR_CLK	I	Clocking signal for ULPI PHY interface.		

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 16-2](#).

Table 16-2. USB Interface Memory Map

Offset	Register	Access	Reset	Section/Page
USB DR Controller Registers				
USB DR Controller—Block Base Address 0x2_3000				
0x000–0x0FF	Reserved, should be cleared	—	—	—
0x100	CAPLENGTH—Capability register length	R	0x40	16.3.1.1/16-6
0x102	HCIVERSION—Host interface version number ¹	R	0x0100	16.3.1.2/16-7
0x104	HCSPARAMS—Host controller structural parameters ¹	R	0x0001_0011	16.3.1.3/16-7
0x108	HCCPARAMS—Host controller capability parameters ¹	R	0x0000_0006	16.3.1.4/16-8
0x120	DCIVERSION—Device interface version number	R	0x0001	16.3.1.5/16-9
0x124	DCCPARAMS—Device controller parameters	R	0x0000_0183	16.3.1.6/16-9
0x140	USBCMD—USB command	Mixed	0x0008_0000	16.3.2.1/16-10
0x144	USBSTS—USB status	Mixed	0x0000_0000	16.3.2.2/16-13
0x148	USBINTR—USB interrupt enable	R/W	0x0000_0000	16.3.2.3/16-15
0x14C	FRINDEX—USB frame index	R/W	All zeros	16.3.2.4/16-16
0x154	PERIODICLISTBASE—Frame list base address ¹	R/W	All zeros	16.3.2.6/16-18
	DEVICEADDR—USB device address	R/W	0x0000_0000	16.3.2.7/16-18
0x158	ASYNCLISTADDR—Next asynchronous list addr (host mode) ¹	R/W	0x0000_0000	16.3.2.8/16-19
	ENDPOINTLISTADDR—Address at endpoint list (device mode)	R/W	0x0000_0000	16.3.2.9/16-20

Table 16-2. USB Interface Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x160	BURSTSIZE—Programmable burst size	R/W	0x0000_1010	16.3.2.10/16-20
0x164	TXFILLTUNING—Host TT transmit pre-buffer packet tuning	R/W	0x0000_0000	16.3.2.11/16-21
0x170	ULPI VIEWPORT—ULPI Register Access	Mixed	0x0000_0000	16.3.2.12/16-22
0x180	CONFIGFLAG—Configured flag register	R	0x0000_0001	16.3.2.13/16-24
0x184	PORTSC—Port status/control	Mixed	0x1000_0000	16.3.2.14/16-24
0x1A4	OTGSC—On-The-Go status and control ¹	Mixed	0x200C_0000	16.3.2.15/16-29
0x1A8	USBMODE—USB device mode	R/W	0x0000_0000	16.3.2.16/16-32
0x1AC	ENDPTSETUPSTAT—Endpoint setup status	w1c	0x0000_0000	16.3.2.17/16-33
0x1B0	ENDPOINTPRIME—Endpoint initialization	R/W	0x0000_0000	16.3.2.18/16-33
0x1B4	ENDPTFLUSH—Endpoint de-initialize	R/W	0x0000_0000	16.3.2.19/16-34
0x1B8	ENDPTSTATUS—Endpoint status	R	0x0000_0000	16.3.2.20/16-35
0x1BC	ENDPTCOMPLETE—Endpoint complete	w1c	0x0000_0000	16.3.2.21/16-35
0x1C0	ENDPTCTRL0—Endpoint control 0	Mixed	0x0080_0080	16.3.2.22/16-36
0x1C4	ENDPTCTRL1—Endpoint control 1	R/W	0x0000_0000	16.3.2.23/16-37
0x1C8	ENDPTCTRL2—Endpoint control 2	R/W	0x0000_0000	16.3.2.23/16-37
0x1CA–0x1D4	Reserved	—	—	—
0x400	SNOOP1—Snoop 1	R/W	0x0000_0000	16.3.2.24/16-39
0x404	SNOOP2—Snoop 2	R/W	0x0000_0000	16.3.2.24/16-39
0x408	AGE_CNT_THRESH—Age count threshold	R/W	0x0000_0000	16.3.2.25/16-40
0x40C	PRI_CTRL—Priority control	R/W	0x0000_0000	16.3.2.26/16-41
0x410	SI_CTRL—System interface control	R/W	0x0000_0000	16.3.2.27/16-42
0x500	CONTROL—Control	R/W	0x0000_0000	16.3.2.28/16-43
0x504–0xFFFF	Reserved, should be cleared	—	—	—

¹ This register has separate functions for the host and device operation; the host function is listed first in the table.

The following sections provide details about the registers in the USB memory map.

NOTE

Memory may be viewed from either a big-endian or little-endian byte ordering perspective depending on the processor configuration. In big-endian mode, the most-significant byte of word 0 is located at address 0 and the least-significant byte of word 0 is located at address 3. In little-endian mode, the least-significant byte of word 0 is located at address 0 and the most-significant byte of word 0 is located at address 3. Within registers, bits are numbered within a word starting with bit 31 as the most-significant bit. By convention USB registers use little-endian byte ordering. In the USB DR module, these are the registers from offsets 0x00 to 0x1FF. The registers associated with the internal system interface (0x400 and above) use big-endian byte ordering.

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 16-2 shows CAPLENGTH.

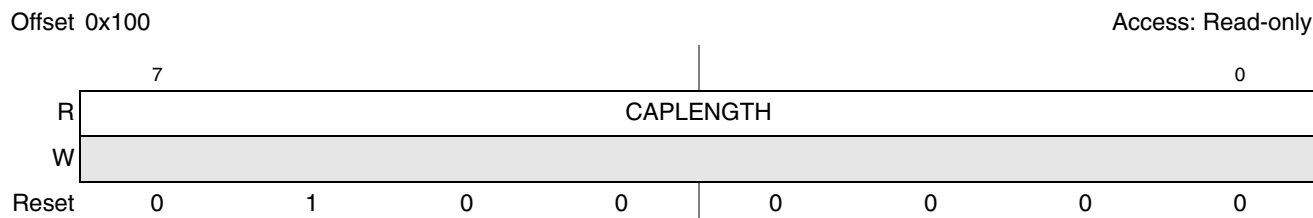


Figure 16-2. Capability Registers Length (CAPLENGTH)

Table 16-3 provides bit descriptions for the CAPLENGTH register.

Table 16-3. 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 16-3](#) shows the HCIVERSION register.

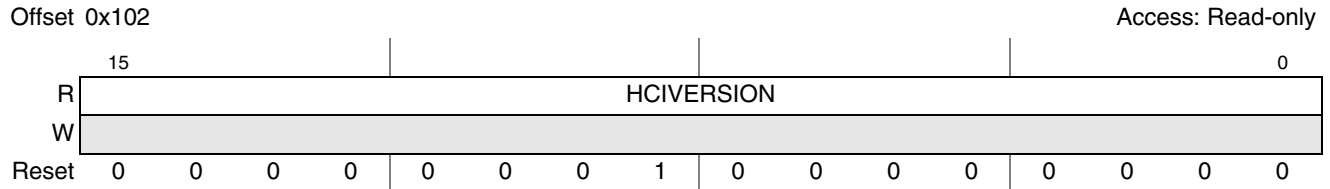


Figure 16-3. Host Controller Interface Version (HCIVERSION)

[Table 16-4](#) provides bit descriptions for the HCIVERSION register.

Table 16-4. HCIVERSION Register Field Descriptions

Bits	Name	Description
15–0	HCIVERSION	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 16-4](#) shows the HCSPARAMS register.

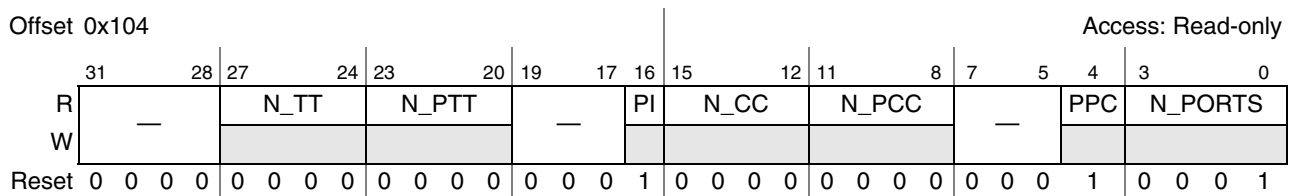


Figure 16-4. Host Controller Structural Parameters (HCSPARAMS)

[Table 16-5](#) provides bit descriptions for the HCSPARAMS register.

Table 16-5. HCSPARAMS Register Field Descriptions

Bits	Name	Description
31–28	—	Reserved, should be cleared.
27–24	N_TT	Number of transaction translators. This is a non-EHCI field. This field indicates the number of embedded transaction translators associated the module. Always 1. See Section 16.9.1, “Embedded Transaction Translator Function.”
23–20	N_PTT	Ports per transaction translator. This is a non-EHCI field. The number of ports assigned to each transaction translator. This is equal to N_PORTS.
19–17	—	Reserved, should be cleared.
16	PI	Port indicators. Indicates whether the ports support port indicator control. Always 1. 1 The port status and control registers include a R/W field for controlling the state of the port indicator.

Table 16-5. HCSPARAMS Register Field Descriptions (continued)

Bits	Name	Description
15–12	N_CC	Number of companion controllers associated with the DR controller. Always 0.
11–8	N_PCC	Number ports per CC. This field indicates the number of ports supported per internal companion controller. Always 0.
7–5	—	Reserved, should be cleared.
4	PPC	Power port control. Indicates whether the host controller supports port power control. Always 1. 1 Ports have power port switches.
3–0	N_PORTS	Number of ports. Number of physical downstream ports implemented for host applications. The value of this field determines how many port registers are addressable in the operational register. Always 1.

16.3.1.4 Host Controller Capability Parameters (HCCPARAMS)

HCCPARAMS identifies multiple mode control (time-base bit functionality) addressing capability. Figure 16-5 shows the HCCPARAMS register.

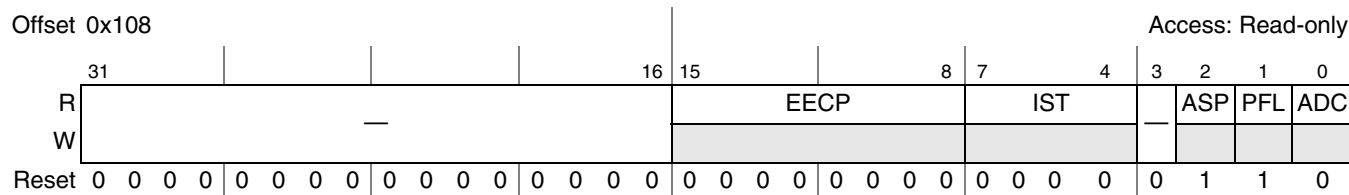

Figure 16-5. Host Control Capability Parameters (HCCPARAMS)

Table 16-6 provides bit descriptions for the HCCPARAMS register.

Table 16-6. HCCPARAMS Register Field Descriptions

Bits	Name	Description
31–16	—	Reserved, should be cleared.
15–8	EECP	EHCI extended capabilities pointer. Indicates the existence of a capabilities list. A value of 0x00 indicates no extended capabilities are implemented.
7–4	IST	Isochronous scheduling threshold. Indicates, relative to the current position of the executing host controller, where software can reliably update the isochronous schedule. When bit 7 is zero, the value of the least significant 3 bits indicates the number of microframes a host controller can hold a set of isochronous data structures (one or more) before flushing the state. When bit 7 is a one, then host software assumes the host controller may cache an isochronous data structure for an entire frame. This field is always 0.
3	—	Reserved, should be cleared.
2	ASP	Asynchronous schedule park capability. Indicates whether the USB DR module supports the park feature for high-speed queue heads in the asynchronous schedule. The feature can be disabled or enabled and set to a specific level using the asynchronous schedule park mode enable and asynchronous schedule park mode count fields in the USBCMD register. This field is always 1 (park feature supported).

Table 16-6. HCCPARAMS Register Field Descriptions (continued)

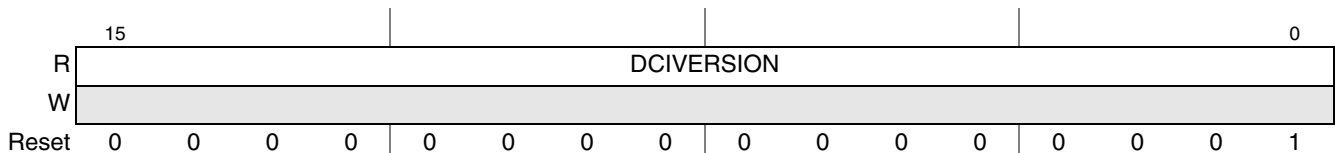
Bits	Name	Description
1	PFL	Programmable frame list flag. Indicates whether system software can specify and use a frame list length less than 1024 elements. Frame list size is configured via the USBCMD register frame list size field. The frame list must always be aligned on a 4-K page boundary. This requirement ensures that the frame list is always physically contiguous. This field is always 1.
0	ADC	64-bit addressing capability. Always 0; 64-bit addressing is not supported. 0 Data structures use 32-bit address memory pointers

16.3.1.5 Device Controller Interface Version (DCIVERSION)—Non-EHCI

This register is not defined in the EHCI specification. DCIVERSION is a two-byte register containing a BCD encoding of the device controller interface. The most-significant byte of the register represents a major revision and the least-significant byte is the minor revision. [Figure 16-6](#) shows the DCIVERSION register.

Offset 0x120

Access: Read-only


Figure 16-6. Device Interface Version (DCIVERSION)

[Table 16-7](#) provides bit descriptions for the DCIVERSION register.

Table 16-7. DCIVERSION Register Field Descriptions

Bits	Name	Description
15–0	DCIVERSION	Device interface revision number.

16.3.1.6 Device Controller Capability Parameters (DCCPARAMS)—Non-EHCI

This register is not defined in the EHCI specification. This register describes the overall host/device capability of the DR module. [Figure 16-7](#) shows the DCCPARAMS register.

Offset 0x124

Access: Read-only

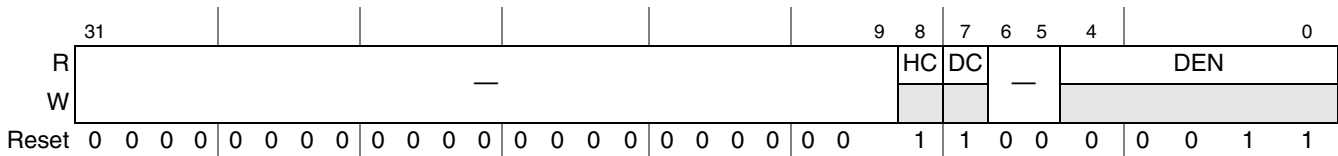

Figure 16-7. Device Control Capability Parameters (DCCPARAMS)

Table 16-8 provides bit descriptions for the DCCPARAMS register.

Table 16-8. DCCPARAMS Register Field Descriptions

Bits	Name	Description
31-9	—	Reserved, should be cleared.
8	HC	Host capable. Always 1, indicating the USB DR controller can operate as an EHCI compatible USB 2.0 host
7	DC	Device capable. Always 1, indicating the USB DR controller can operate as an USB 2.0 device. 1 Device capability. 0 No device capability (host only).
6-5	—	Reserved, should be cleared.
4-0	DEN	Device endpoint number. Indicates the number of endpoints built into the device controller. Always 0x3.

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)

Figure 16-8 shows the USB command register. The module executes the command indicated in this register.

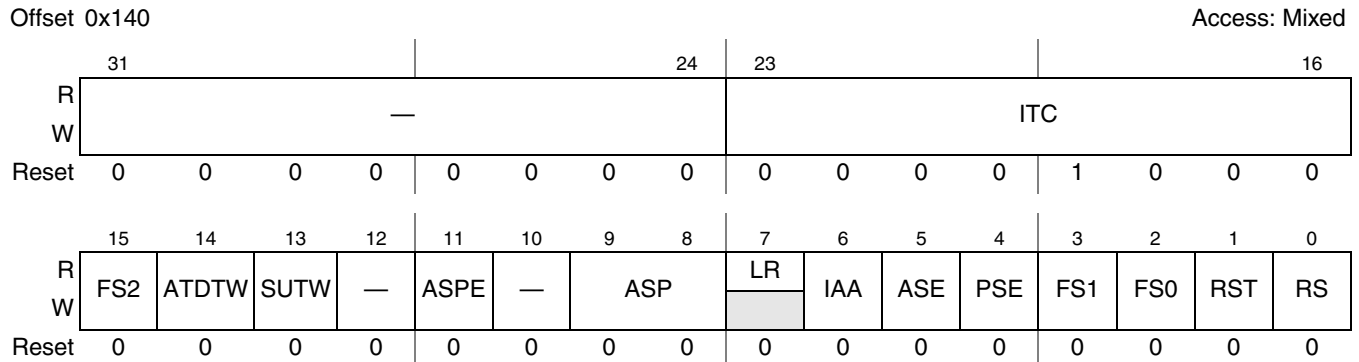


Figure 16-8. USB Command Register (USBCMD)

Table 16-9 provides bit descriptions for the USBCMD register.

Table 16-9. USBCMD Register Field Descriptions

Bits	Name	Description
31–24	—	Reserved, should be cleared.
23–16	ITC	Interrupt threshold control. The system software uses this field to set the maximum rate at which the USB DR module issues interrupts. ITC contains the maximum interrupt interval measured in microframes. Valid values are shown below. 0x00 Immediate (no threshold) 0x01 1 microframe 0x02 2 microframes 0x04 4 microframes 0x08 8 microframes 0x10 16 microframes 0x20 32 microframes 0x40 40 microframes
15	FS2	See bits 3–2 below. This is a non-EHCI bit.
14	ATDTW	Add dTD TripWire. This is a non-EHCI bit. Used as a semaphore when a dTD is added to an active (primed) endpoint. This bit is set and cleared by software. This bit shall also be cleared by hardware when its state machine is in 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.”
13	SUTW	Setup tripwire. This is a non-EHCI bit. Used as a semaphore when the 8 bytes of setup data read extracted from a QH by the DCD. If the setup lockout mode is off (See USBMODE) then there exists a hazard when new setup data arrives and the DCD is copying setup from the QH for a previous setup packet. This bit is set and cleared by software and will be cleared by hardware when a hazard exists. More information on the use of this bit is described in Section 16.9.2, “Device Operation.”
12	—	Reserved, should be cleared.
11	ASPE	Asynchronous schedule park mode enable. This bit defaults to a 1 and is R/W. Software uses this bit to enable or disable park mode. 0 Disabled 1 Enabled
10	—	Reserved, should be cleared.
9–8	ASP	Asynchronous schedule park mode count. This field defaults to 0x3 and is R/W. It contains a count of the number of successive transactions the host controller is allowed to execute from a high-speed queue head on the Asynchronous schedule before continuing traversal of the Asynchronous schedule. Valid values are 0x1H to 0x3H. Software must not write a zero to this field when ASPE is set as this results in undefined behavior.
7	LR	Light host/device controller reset (OPTIONAL). Not implemented. Always 0.
6	IAA	Interrupt on async advance doorbell. Used as a doorbell by software to tell the USB DR controller to issue an interrupt the next time it advances asynchronous schedule. Software must write a 1 to this bit to ring the doorbell. When the controller has evicted all appropriate cached schedule states, it sets USBSTS[AAI]. If USBINTR[AAE] is set, the host controller asserts an interrupt at the next interrupt threshold. The controller clears this bit after it has set USBSTS[AAI]. Software should not set this bit when the asynchronous schedule is inactive. Doing so yields undefined results. This bit is only used in host mode. Setting this bit when the USB DR module is in device mode is selected results in undefined results.

Table 16-9. USBCMD Register Field Descriptions (continued)

Bits	Name	Description
5	ASE	Asynchronous schedule enable. Controls whether the controller skips processing the asynchronous schedule. Only used in host mode. 0 Do not process the asynchronous schedule 1 Use the ASYNCLISTADDR register to access the asynchronous schedule.
4	PSE	Periodic schedule enable. Controls whether the controller skips processing the periodic schedule. Only used in host mode. 0 Do not process the periodic schedule. 1 Use the PERIODICLISTBASE register to access the periodic schedule.
3–2	FS	Frame list size. Together with bit 15 these bits make the FS[2:0] field. This field is read/write only if programmable frame list flag in the HCCPARAMS registers is set to 1. This field specifies the size of the frame list that controls which bits in FRINDEX should be used for the frame list current index. Only used in host mode. Note that values below 256 elements are not defined in the EHCI specification. 000 1024 elements (4096 bytes) 001 512 elements (2048 bytes) 010 256 elements (1024 bytes) 011 128 elements (512 bytes) 100 64 elements (256 bytes) 101 32 elements (128 bytes) 110 16 elements (64 bytes) 111 8 elements (32 bytes)
1	RST	Controller reset. Software uses this bit to reset the controller. This bit is cleared by the controller when the reset process is complete. Software cannot terminate the reset process early by writing a zero to this register. Host mode: <ul style="list-style-type: none"> When software sets this bit, the host controller resets its internal pipelines, timers, counters, state machines etc. to their initial value. Any transaction currently in progress on USB is immediately terminated. A USB reset is not driven on downstream ports. Software should not set this bit when USBSTS[HCH] is a zero. Attempting to reset an actively running host controller results in undefined behavior. Device mode: <ul style="list-style-type: none"> When software sets this bit, the USB DR controller resets its internal pipelines, timers, counters, state machines etc. to their initial value. Any transaction currently in progress on USB is immediately terminated. Writing a one to this bit in device mode is not recommended.
0	RS	Run/Stop. Host mode: <ul style="list-style-type: none"> When this bit is set, the controller proceeds with the execution of the schedule. The controller continues execution as long as this bit is set. When this bit is set to 0, the host controller completes the current transaction on the USB and then halts. The USBSTS[HCH] bit indicates when the USB DR controller has finished the transaction and has entered the stopped state. Software should not write a one to this field unless the controller is in the halted state (that is, USBSTS[HCH] is a one). Device mode: <ul style="list-style-type: none"> Setting this bit causes the USB DR controller to enable a pull-up on D+ and initiate an attach event. This control bit is not directly connected to the pull-up enable, as the pull-up is disabled upon transitioning into high-speed mode. Software should use this bit to prevent an attach event before the controller has been properly initialized. Clearing this bit causes a detach event. 0 Stop 1 Run

16.3.2.2 USB Status Register (USBSTS)

Figure 16-9 shows the USB status register, which indicates various states of the USB DR module and any pending interrupts. This register does not indicate status resulting from a transaction on the serial bus. Software clears certain bits in this register by writing a 1 to them (indicated by a w1c in the bit's W cell).

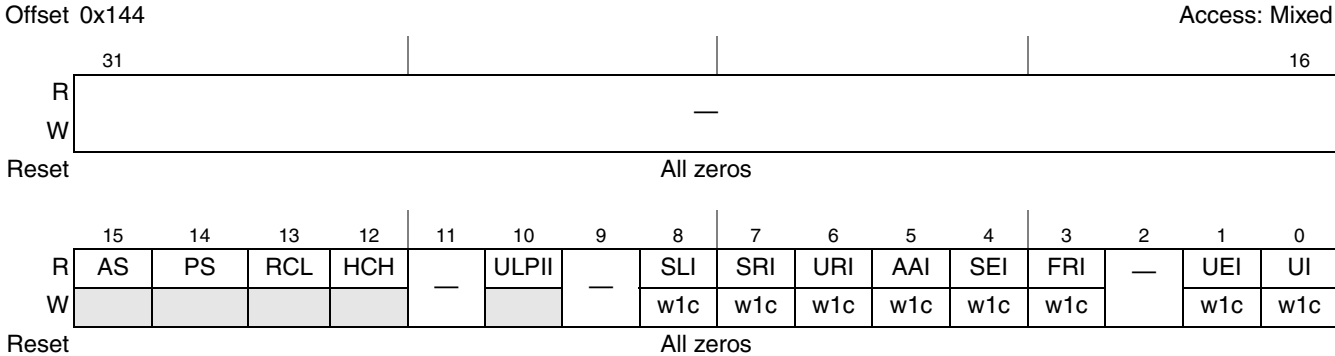


Figure 16-9. USB Status Register (USBSTS)

Table 16-10 shows the USBSTS register field descriptions.

Table 16-10. USBSTS Register Field Descriptions

Bits	Name	Description
31–16	—	Reserved, should be cleared.
15	AS	Asynchronous schedule status. Reports the current real status of the asynchronous schedule. The USB DR controller is not required to immediately disable or enable the asynchronous schedule when software transitions USBCMD[ASE]. When this bit and USBCMD[ASE] have the same value, the asynchronous schedule is either enabled (1) or disabled (0). Only used in host mode. 0 Disabled 1 Enabled
14	PS	Periodic schedule status. Reports the current real status of the periodic schedule. The USB DR controller is not required to immediately disable or enable the periodic schedule when software transitions USBCMD[PSE]. When this bit and USBCMD[PSE] have the same value, the periodic schedule is either enabled (1) or disabled (0). Only used in host mode. 0 Disabled 1 Enabled
13	RCL	Reclamation. Used to detect an empty asynchronous schedule. Only used by the host mode. 0 Non-empty asynchronous schedule 1 Empty asynchronous schedule
12	HCH	HC halted. This bit is a zero whenever USBCMD[RS] is a one. The USB DR controller sets this bit to one after it has stopped executing because of USBCMD[RS] being cleared, either by software or by the host controller hardware (for example, internal error). Only used in host mode. 0 Running 1 Halted
11	—	Reserved, should be cleared.
10	ULPII	ULPI interrupt. An event completion to the viewport register sets this bit. If the ULPI enables the USBINTR[ULPIE] to be set, the USB interrupt (UI) occurs.
9	—	Reserved, should be cleared.

Table 16-10. USBSTS Register Field Descriptions (continued)

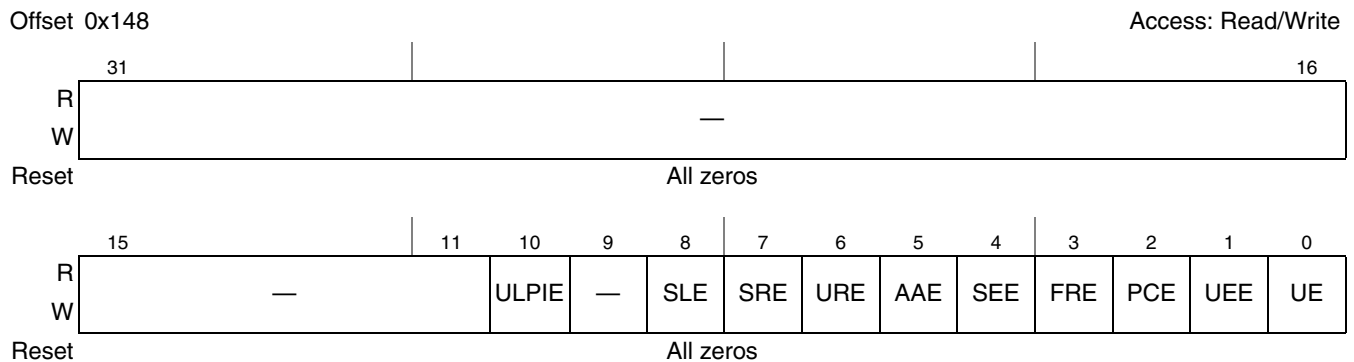
Bits	Name	Description
8	SLI	DCSuspend. This is a non-EHCI bit. When a device controller enters a suspend state from an active state, this bit is set. The device controller clears the bit upon exiting from a suspend state. Only used by the device controller. 0 Active 1 Suspended
7	SRI	Host mode: <ul style="list-style-type: none"> This is a non-EHCI status bit. In host mode, this bit is set every 125 us, provided the PHY clock is present and running (for example, the port is NOT suspended), and can be used by the host controller driver as a time base. Device mode: <ul style="list-style-type: none"> SOF received. When the USB DR controller detects a Start Of (Micro)Frame, this bit is set. When a SOF is extremely late, the DR controller automatically sets this bit to indicate that an SOF was expected. Therefore, this bit is set roughly every 1 msec in device FS mode and every 125 us in HS mode and is synchronized to the actual SOF that is received. Because the controller is initialized to FS before connect, this bit is set at an interval of 1 msec during the prelude to the connect and chirp. Software writes a 1 to this bit to clear it.
6	URI	USB reset received. This is a non-EHCI bit. When the USB DR controller detects a USB reset and enters the default state, this bit is set. Software can write a 1 to this bit to clear the USB reset received status bit. Only used by the device mode. 0 No reset received 1 Reset received
5	AAI	Interrupt on async advance. System software can force the controller to issue an interrupt the next time the USB DR controller advances the asynchronous schedule by writing a one to USBCMD[IAA]. This status bit indicates the assertion of that interrupt source. Only used by the host mode. 0 No async advance interrupt 1 Async advance interrupt
4	SEI	System error. This bit is set whenever an error is detected on the system bus. If USBINTR[SEE] is set, an interrupt is generated. The interrupt and status bits remain asserted until cleared by writing a 1 to this bit. Additionally, when in host mode, USBCMD[RS] is cleared, effectively disabling the USB DR controller. For the USB DR controller in device mode, an interrupt is generated, but no other action is taken. 0 Normal operation 1 Error
3	FRI	Frame list rollover. The controller sets this bit to a one when the frame list index rolls over from its maximum value to zero. The exact value at which the rollover occurs depends on the frame list size. For example. If the frame list size (as programmed in USBCMD[FS]) is 1024, FRINDEX rolls over every time FRINDEX [1 3] toggles. Similarly, if the size is 512, the USB DR controller sets this bit to a one every time FHINDEX [12] toggles. Only used by the host mode.
2	—	Reserved

Table 16-10. USBSTS Register Field Descriptions (continued)

Bits	Name	Description
1	UEI (USBERRINT)	USB error interrupt (USBERRINT). When completion of a USB transaction results in an error condition, this bit is set by the controller. This bit is set along with the UI, if the TD on which the error interrupt occurred also had its interrupt on complete (IOC) bit set. See Section 4.15.1 in EHCI for a complete list of host error interrupt conditions. Also see Table 16-90 in this chapter for more information on device error matrix. For the USB DR controller in device mode, only resume signaling is detected, all others are ignored. 0 No error 1 Error detected
0	UI (USBINT)	USB interrupt (USBINT). This bit is set by the controller when the cause of an interrupt is a completion of a USB transaction where the transfer descriptor (TD) has an interrupt on complete (IOC) bit set. This bit is also set by the controller when a short packet is detected. A short packet is when the actual number of bytes received was less than the expected number of bytes.

16.3.2.3 USB Interrupt Enable Register (USBINTR)

The interrupts to software are enabled with the USB interrupt enable register, shown in [Figure 16-10](#). An interrupt is generated when a bit is set and the corresponding interrupt is active. The USB status register (USBSTS) still shows interrupt sources even if they are disabled by the USBINTR register, allowing polling of interrupt events by the software.


Figure 16-10. USB Interrupt Enable (USBINTR)

[Table 16-11](#) shows the USBINTR register field descriptions.

Table 16-11. USBINTR Register Field Descriptions

Bits	Name	Description
31–11	—	Reserved, should be cleared.
10	ULPIE	ULPI interrupt enable. An event completion to the viewport register sets the USBSTS[ULPII]. If the ULPI enables ULPIE bit to be set, then the USBINT (USBSTS[UI]) occurs. 0 Disable 1 Enable
9	—	Reserved, should be cleared.

Table 16-11. USBINTR Register Field Descriptions (continued)

Bits	Name	Description
8	SLE	Sleep enable. This is a non-EHCI bit. When this bit is a one, and USBSTS[SLI] transitions, the USB DR controller issues an interrupt. The interrupt is acknowledged by software writing a one to USBSTS[SLI]. Only used in device mode. 0 Disable 1 Enable
7	SRE	SOF received enable. This is a non-EHCI bit. When this bit is a one, and USBSTS[SRI] is a one, the controller issues an interrupt. The interrupt is acknowledged by software clearing USBSTS[SRI]. 0 Disable 1 Enable
6	URE	USB reset enable. This is a non-EHCI bit. When this bit is a one, USBSTS[URI] is a one, the device controller issues an interrupt. The interrupt is acknowledged by software clearing USBSTS[URI] bit. Only used in device mode. 0 Disable 1 Enable
5	AAE	Interrupt on async advance enable. When this bit is a one, and USBSTS[AAI] is a one, the controller issues an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing USBSTS[AAI]. Only used in host mode. 0 Disable 1 Enable
4	SEE	System error enable. When this bit is a one, and USBSTS[SEI] is a one, the controller issues an interrupt. The interrupt is acknowledged by software clearing USBSTS[SEI]. 0 Disable 1 Enable
3	FRE	Frame list rollover enable. When this bit is a one, and USBSTS[FRI] is a one, the controller issues an interrupt. The interrupt is acknowledged by software clearing USBSTS[FRI]. Only used by the host mode. 0 Disable 1 Enable
2	PCE	Port change detect enable.
1	UEE	USB error interrupt enable. When this bit is a one, and USBSTS[UEI] is a one, the controller issues an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing USBSTS[UEI]. 0 Disable 1 Enable
0	UE	USB interrupt enable. When this bit is a one, and USBSTS[UI] is a one, the DR controller issues an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing USBSTS[UI]. 0 Disable 1 Enable

16.3.2.4 Frame Index Register (FRINDEX)

In host mode, the frame index register is used by the controller to index the periodic frame list. The register updates every 125 microseconds (once each microframe). Bits N–3 are used to select a particular entry in the periodic frame list during periodic schedule execution. The number of bits used for the index depends on the size of the frame list as set by system software in USBCMD[FS].

This register must be written as a DWORD. Byte writes produce undefined results. This register cannot be written unless the USB DR controller is in the Halted state as indicated by the USBSTS[HCH]. A write to

this register while USBCMD[RS] is set produces undefined results. Writes to this register also affect the SOF value.

In device mode, this register is read-only and the USB DR controller updates the FRINDEX[13–3] register from the frame number indicated by the SOF marker. Whenever a SOF is received by the USB bus, FRINDEX[13–3] is checked against the SOF marker. If FRINDEX[13–3] is different from the SOF marker, FRINDEX[13–3] is set to the SOF value and FRINDEX[2–0] is cleared (that is, SOF for 1 msec frame). If FRINDEX[13–3] is equal to the SOF value, FRINDEX[2–0] is incremented (that is, SOF for 125- μ sec microframe).

Figure 16-11 shows the USB frame index register.

Offset 0x14C

Access: Read/Write

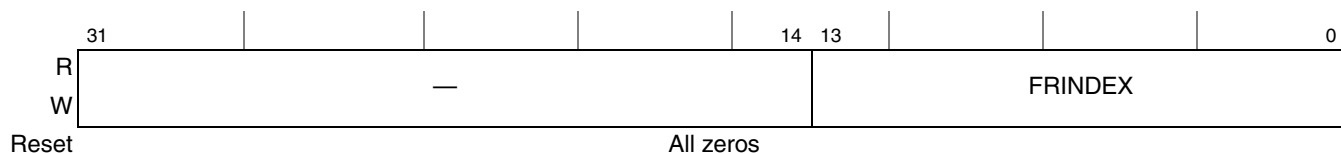


Figure 16-11. USB Frame Index (FRINDEX)

Table 16-12 shows the FRINDEX register field descriptions.

Table 16-12. FRINDEX Register Field Descriptions

Bits	Name	Description
31–14	—	Reserved, should be cleared.
13–0	FRINDEX	Frame index. The value in this register increments at the end of each time frame (for example, microframe). Bits N–3 are used for the Frame List current index. This means that each location of the frame list is accessed 8 times (frames or microframes) before moving to the next index. In device mode, the value is the current frame number of the last frame transmitted. It is not used as an index. In either mode, bits 2–0 indicate the current microframe.

Table 16-13 illustrates values of N based on the value of the Frame List Size in the USBCMD register, when used in host mode.

Table 16-13. FRINDEX N Values

USBCMD[FS]	Frame List Size	FRINDEX N value
000	1024 elements (4096 bytes)	12
001	512 elements (2048 bytes)	11
010	256 elements (1024 bytes)	10
011	128 elements (512 bytes)	9
100	64 elements (256 bytes)	8
101	32 elements (128 bytes)	7
110	16 elements (64 bytes)	6
111	8 elements (32 bytes)	5

16.3.2.5 Control Data Structure Segment Register (CTRLDSSEGMENT)

The CTRLDSSEGMENT register is not implemented on MPC8309.

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 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 \(DEVICEADDR\)—Non-EHCI,”](#) for more information.

Figure 16-12 shows the periodic frame list base address register.



Figure 16-12. Periodic Frame List Base Address (PERIODICLISTBASE)

Table 16-14 shows the periodic frame list base address register field descriptions.

Table 16-14. PERIODICLISTBASE Register Field Descriptions

Bits	Name	Description
31–12	PERBASE	Base address. Correspond to memory address signal [31:12]. Only used in the host mode.
11–0	—	Reserved, should be cleared.

16.3.2.7 Device Address Register (DEVICEADDR)—Non-EHCI

The device address register is not defined in the EHCI specification. In device mode, the upper seven bits of this register represent the device address. After any controller reset or a USB reset, the device address is set to the default address (0). The default address will match all incoming addresses. Software shall reprogram the address after receiving a SET_ADDRESS descriptor.

Note that this register is shared between the host and device mode functions. In device mode, it is the DEVICEADDR register; in host mode, it is the PERIODICLISTBASE register. See [Section 16.3.2.6, “Periodic Frame List Base Address Register \(PERIODICLISTBASE\),”](#) for more information.

Figure 16-12 shows the device address register.

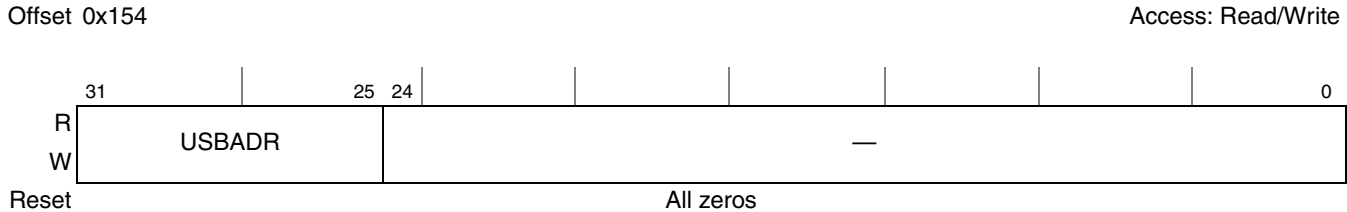


Figure 16-13. Device Address (DEVICEADDR)

Table 16-15 shows the device address register field descriptions.

Table 16-15. DEVICEADDR Register Field Descriptions

Bits	Name	Description
31–25	USBADR	Device address. This field corresponds to the USB device address.
24–0	—	Reserved, should be cleared.

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 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)—Non-EHCI,” for more information.

Figure 16-14 shows the current asynchronous list address register.

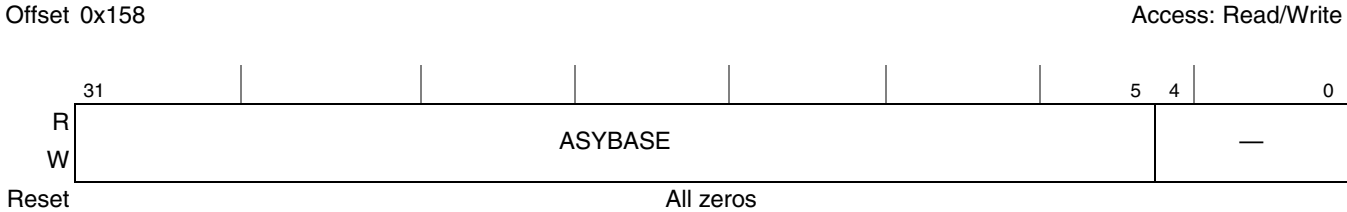


Figure 16-14. Current Asynchronous List Address (ASYNCLISTADDR)

Table 16-16 describes the current asynchronous list address register.

Table 16-16. ASYNCLISTADDR Register Field Descriptions

Bits	Name	Description
31–5	ASYBASE	Link pointer low (LPL). These bits correspond to memory address signals [31:5]. This field may only reference a queue head (QH). Only used by the host controller.
4–0	—	Reserved, should be cleared.

16.3.2.9 Endpoint List Address Register (ENDPOINTLISTADDR)—Non-EHCI

The endpoint list address register is not defined in the EHCI specification. In device mode, this register contains the address of the top of the endpoint list in system memory. Bits 10–0 of this register cannot be modified by the system software and always return zeros when read. The memory structure referenced by this physical memory pointer is assumed to be 64-bytes. The queue head is actually a 48-byte structure, but must be aligned on 64-byte boundary. However, the ENDPOINTLISTADDR[EPBASE] has a granularity of 2 Kbytes, so in practice the queue head should be 2-Kbyte aligned.

Note that this register is shared between the host and device mode functions. In device mode, it is the ENDPOINTLISTADDR register; in host mode, it is the ASYNCLISTADDR register. See [Section 16.3.2.8, “Current Asynchronous List Address Register \(ASYNCLISTADDR\),”](#) for more information.

Figure 16-15 shows the endpoint list address register.

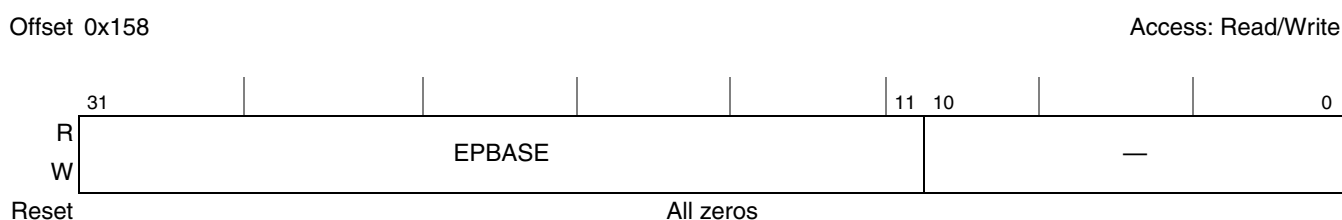


Figure 16-15. Endpoint List Address (ENDPOINTLISTADDR)

Table 16-17 describes the endpoint list address register fields.

Table 16-17. ENDPOINTLISTADDR Register Field Descriptions

Bits	Name	Description
31–11	EPBASE	Endpoint list address. Address of the top of the endpoint list.
10–0	—	Reserved, should be cleared.

16.3.2.10 Master Interface Data Burst Size Register (BURSTSIZE)—Non-EHCI

The master interface data burst size register, shown in Figure 16-16, is not defined in the EHCI specification. This register is used to control and dynamically change the burst size used during data movement on the initiator (master) interface.

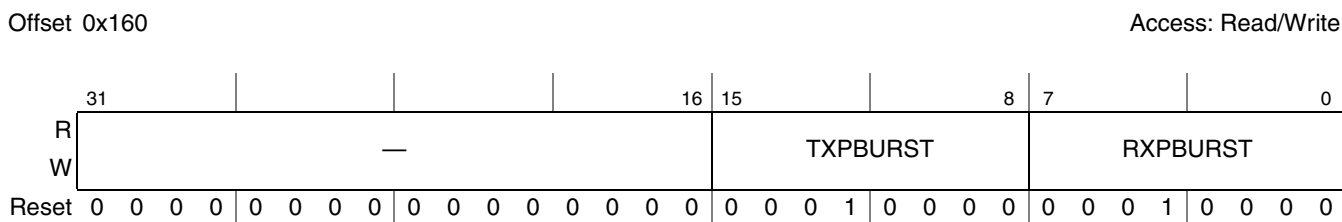


Figure 16-16. Master Interface Data Burst Size (BURSTSIZE)

Table 16-18 describes the master interface data burst size register fields.

Table 16-18. BURSTSIZE Register Field Descriptions

Bits	Name	Description
31–16	—	Reserved, should be cleared.
15–8	TXPBURST	Programmable TX burst length. This register represents the maximum length of a burst in 32-bit words while moving data from system memory to the USB bus. Must not be set to greater than 16.
7–0	RXPBURST	Programmable RX burst length. This register represents the maximum length of a burst in 32-bit words while moving data from the USB bus to system memory. Must not be set to greater than 16.

16.3.2.11 Transmit FIFO Tuning Controls Register (TXFILLTUNING)—Non-EHCI

The transmit FIFO tuning controls register, shown in Figure 16-17, is not defined in the EHCI specification. This register is used to control and dynamically change the burst size used during data movement on device DMA transfers. It is only used in host mode.

The fields in this register control performance tuning associated with how the USB DR module posts data to the TX latency FIFO before moving the data onto the USB bus. The specific areas of performance include the how much data to post into the FIFO and an estimate for how long that operation should take in the target system.

Definitions:

T_0 = Standard packet overhead

T_I = Time to send data payload

T_s = Total Packet Flight Time (send-only) packet ($T_s = T_0 + T_I$)

T_{ff} = Time to fetch packet into TX FIFO up to specified level.

T_p = Total Packet Time (fetch and send) packet ($T_p = T_{ff} + T_s$)

Upon discovery of a transmit (OUT/SETUP) packet in the data structures, host controller checks to ensure T_p remains before the end of the [micro]frame. If so it proceeds to pre-fill the TX FIFO. If at any time during the pre-fill operation the time remaining the [micro]frame is $< T_s$, then the packet attempt ceases and the packet is tried at a later time. Although this is not an error condition and the module eventually recovers, a mark is made in the scheduler health counter to note the occurrence of a back-off event. When a back-off event is detected, the partial packet fetched may need to be discarded from the latency buffer to make room for periodic traffic that will begin after the next SOF. Too many back-off events can waste bandwidth and power on the system bus and thus should be minimized (not necessarily eliminated). Back-offs can be minimized with use of the TSCHEALTH (T_{ff}) parameter described below.

Offset 0x164

Access: Read/Write

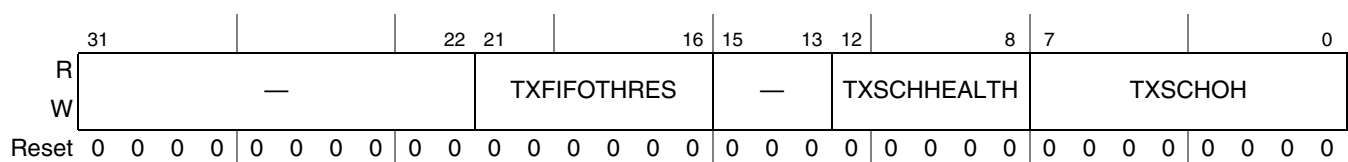


Figure 16-17. Transmit FIFO Tuning Controls (TXFILLTUNING)

Table 16-19 describes the transmit FIFO tuning controls register fields.

Table 16-19. TXFILLTUNING Register Field Descriptions

Bits	Name	Description
31–22	—	Reserved, should be cleared.
21–16	TXFIFOTHRES	FIFO burst threshold. Control the number of data bursts that are posted to the TX latency FIFO in host mode before the packet begins on to the bus. The minimum value is 2 and this value should be as low as possible to maximize USB performance. A higher value can be used in systems with unpredictable latency and/or insufficient bandwidth where the FIFO may underrun because the data transferred from the latency FIFO to USB occurs before it can be replenished from system memory. This value is ignored if USBMODE[SDIS] (stream disable bit) is set. When USBMODE[SDIS] is set, the host controller behaves as if TXFIFOTHRES is set to the maximum value.
15–13	—	Reserved, should be cleared.
12–8	TXSCHHEALTH	Scheduler health counter. Increment when the host controller fails to fill the TX latency FIFO to the level programmed by TXFIFOTHRES before running out of time to send the packet before the next Start-Of-Frame. This health counter measures the number of times this occurs to provide feedback to selecting a proper TXSCHOH. Writing to this register clears the counter and this counter stops counting after reaching the maximum of 31.
7–0	TXSCHOH	Scheduler overhead. These bits add an additional fixed offset to the schedule time estimator described above as T_{ff} . As an approximation, the value chosen for this register should limit the number of back-off events captured in the TXSCHHEALTH to less than 10 per second in a highly utilized bus. Choosing a value that is too high for this register is not desired as it can needlessly reduce USB utilization. The time unit represented in this register is 1.267 μ s when a device is connected in high-speed mode. The time unit represented in this register is 6.333 μ s when a device is connected in low-/full-speed mode. For most applications, TXSCHOH can be set to 4 or less. A good value to begin with is: $TXFIFOTHRES \times (BURSTSIZE \times 4 \text{ bytes-per-word}) \div (40 \times TimeUnit)$, always rounded to the next higher integer. <i>TimeUnit</i> is either 1.267 or 6.333 as noted earlier in this description. For example, if TXFIFOTHRES is 5 and BURSTSIZE is 8, then set TXSCHOH to $5 \times (8 \times 4) \div (40 \times 1.267) = 4$ for a high-speed link. If this value of TXSCHOH results in a TXSCHHEALTH count of 0 per second, try lowering the value by 1 if optimizing performance is desired. If TXSCHHEALTH exceeds 10 per second, try raising the value by 1. If streaming mode is disabled via the USBMODE register, treat TXFIFOTHRES as the maximum value for purposes of the TXSCHOH calculation.

16.3.2.12 ULPI Register Access (ULPI VIEWPORT)

The ULPI register access provides indirect access to the ULPI PHY register set. Although the controller modules perform access to the ULPI PHY register set, there may be extraordinary circumstances where software may need direct access. Be advised that writes to the ULPI through the ULPI viewport can substantially harm standard USB operations. Currently no usage model has been defined where software should need to execute writes directly to the ULPI. Note that executing read operations through the ULPI viewport should have no harmful side effects to standard USB operations. Also note that if the ULPI interface is not enabled, this register will always read zeros.

ULPI VIEWPORT is shown in Figure 16-18.

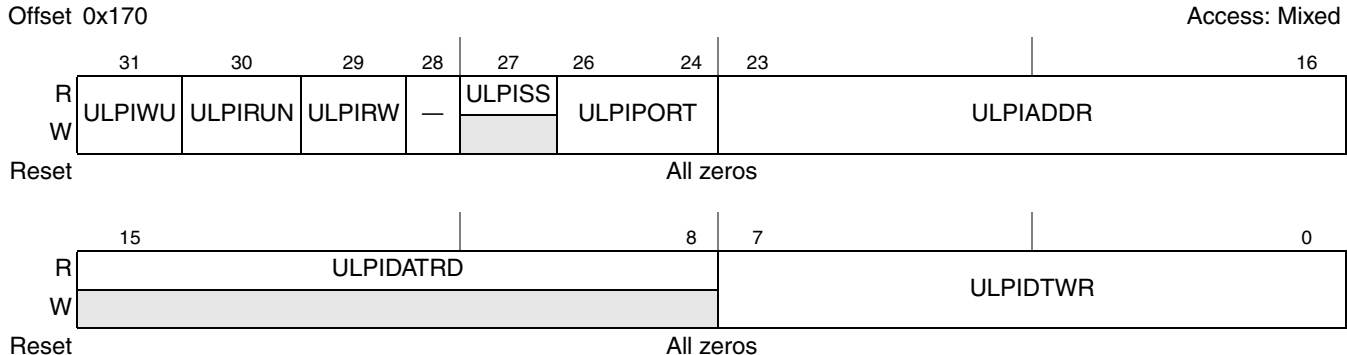


Figure 16-18. ULPI Register Access (ULPI VIEWPORT)

Table 16-20 describes the ULPI register access fields.

Table 16-20. ULPI VIEWPORT Field Descriptions

Bits	Name	Description
31	ULPIWU	ULPI Wake Up. Writing 1 to this bit begins the wakeup operation. This bit automatically transitions to 0 after the wakeup is complete. Once this bit is set, it can not be cleared by software. 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. Otherwise, this field should always remain 0. 0 Any other state (that is carkit and 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 value is 0.
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 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 ULPIS 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.

16.3.2.13 Configure Flag Register (CONFIGFLAG)

This EHCI register is not used in this implementation. A read from this register returns a constant of a 0x0000_0001 to indicate that all port routings default to this host controller.

Figure 16-19 shows the configure flag register.

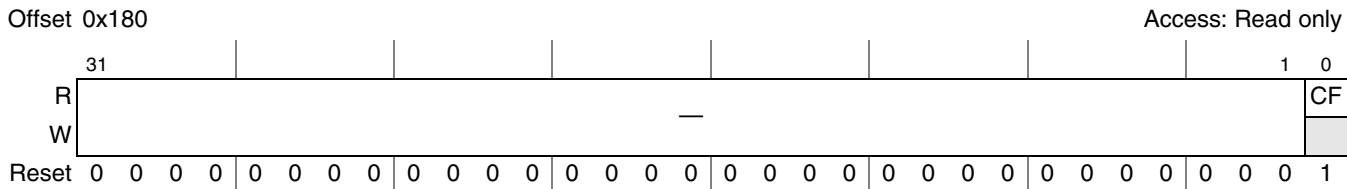


Figure 16-19. Configure Flag Register (CONFIGFLAG)

Table 16-21 describes the configure flag register fields.

Table 16-21. CONFIGFLAG Register Field Descriptions

Bits	Name	Description
31-1	—	Reserved.
0	CF	Configure flag. Always 1 indicating all port routings default to this host.

16.3.2.14 Port Status and Control Register (PORTSC)

The port status and control (PORTSC) register, shown in Figure 16-20, is only reset when power is initially applied or in response to a controller reset. The initial conditions of a port are:

- No device connected
- Port disabled

If the port has port power control, this state remains until software applies power to the port by setting port power to one.

In device mode, the USB DR controller does not support power control. Port control in device mode is only used for status port reset, suspend, and current connect status. It is also used to initiate test mode or

force signaling and allows software to put the PHY into low power suspend mode and disable the PHY clock.

Offset 0x184

Access: Mixed

	31	30	29	28	27	26	25	24	23	22	21	20	19	16		
R	PTS		—	—	PSPD		—	PFSC	PHCD	WKOC	WKDS	WKCN	PTC			
W																
Reset	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PIC		RO	PP	LS		—	PR	SUSP	FPR	OCC	OCA	PEC	PE	CSC	CCS
W											w1c		w1c			w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-20. Port Status and Control (PORTSC)

Table 16-22 describes the PORTSC register fields.

Table 16-22. PORTSC Register Field Descriptions

Bits	Name	Description
31–30	PTS	Port transceiver select. This register bit is used to control which parallel transceiver interface is selected. 00 Reserved 01 Reserved 10 ULPI parallel interface 11 Reserved This bit is not defined in the EHCI specification.
29	—	Reserved, should be cleared
28	—	Reserved
27–26	PSPD	Port speed. This read-only register field indicates the speed at which the port is operating. This bit is not defined in the EHCI specification. 00 Full-speed 01 Low-speed 10 High-speed 11 Undefined
25	—	Reserved, should be cleared
24	PFSC	Port force full-speed connect. Used to disable the chirp sequence that allows the port to identify itself as a HS port. This is useful for testing FS configurations with a HS host, hub or device. 0 Allow the port to identify itself as high speed. 1 Force the port to only connect at full speed. This bit is not defined in the EHCI specification. This bit is for debugging purposes.

Table 16-22. PORTSC Register Field Descriptions (continued)

Bits	Name	Description
23	PHCD	PHY low power suspend. This bit is not defined in the EHCI specification. Host mode: <ul style="list-style-type: none"> 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. Device mode: <ul style="list-style-type: none"> The PHY can be put into low power suspend – when the device is not running (USBCMD[RS] = 0b) or suspend signaling is detected on the USB. Low power suspend is cleared automatically when the resume signaling has been detected or when forcing port resume. 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 USBDR_CLK signals, PHCD must be set and the following registers should not be written: DEVICE_ADDR/PERIODICLISTBASE, PORTSC, ENDPTCTRL0, ENDPTCTRL1, ENDPTCTRL2.
22	WKOC	Wake on over-current enable. Writing this bit to a one enables the port to be sensitive to over-current conditions as wake-up events. This field is zero if Port Power (PP) is zero. This bit is (OTG/host mode only) for use by an external power control circuit.
21	WKDS	Wake on disconnect enable. Writing this bit to a one enables the port to be sensitive to device disconnects as wake-up events. This field is zero if Port Power(PP) is zero or in device mode. This bit is (OTG/host mode only) for use by an external power control circuit.
20	WKCN	Wake on connect enable. Writing this bit to a one enables the port to be sensitive to device connects as wake-up events. This field is zero if Port Power(PP) is zero or in device mode. This bit is (OTG/host mode only) for use by an external power control circuit.
19–16	PTC	Port test control. Any other value than zero indicates that the port is operating in test mode. 0000 Not Enabled 0001 J_STATE 0010 K_STATE 0011 SEQ_NAK 0100 Packet 0101 FORCE_ENABLE 0110–1111 Reserved, should be cleared Refer to Chapter 7 of the USB Specification Revision 2.0 [3] for details on each test mode.
15–14	PIC	Port indicator control. Control the link indicator signals. These signals are valid for host mode only. 00 Off 01 Amber 10 Green 11 Undefined Refer to the USB Specification Revision 2.0 [3] for a description on how these bits are to be used. This field is output from the module on the USB port control signals for use by an external LED driving circuit.
13	RO	Port owner. Unconditionally goes to a 0 when the configured bit in the CONFIGFLAG register makes a 0 to 1 transition. This bit unconditionally goes to 1 whenever the Configured bit is zero. System software uses this field to release ownership of the port to a selected the module (in the event that the attached device is not a high-speed device). Software writes a one to this bit when the attached device is not a high-speed device. A one in this bit means that an internal companion controller owns and controls the port. Port owner hand-off is not implemented in this design, therefore this bit is always 0.

Table 16-22. PORTSC Register Field Descriptions (continued)

Bits	Name	Description
12	PP	<p>Port power. Represents the current setting of the switch (0=off, 1=on). When power is not available on a port (that is, PP equals a 0), the port is non-functional and will not report attaches, detaches, etc. When an over-current condition is detected on a powered port, the PP bit in each affected port is transitioned by the host controller driver from a one to a zero (removing power from the port). This feature is implemented in the host/OTG controller (PPC = 1). In a device-only implementation port power control is not necessary, thus PPC and PP = 0.</p>
11–10	LS	<p>Line status. Reflect the current logical levels of the USB D+ (bit 11) and D– (bit 10) signal lines. The use of line status by the host controller driver is not necessary (unlike EHCI), because the connection of FS and LS is managed by hardware.</p> <p>00 SE0 10 J-state 01 K-state 11 Undefined</p>
9	—	Reserved, should be cleared
8	PR	<p>Port reset.</p> <p>Host mode:</p> <ul style="list-style-type: none"> 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. <p>Device mode:</p> <ul style="list-style-type: none"> This bit is a read only status bit. Device reset from the USB bus is also indicated in the USBSTS register. <p>1 Port is in reset. 0 Port is not in reset. This field is zero if Port Power(PP) is zero.</p>
7	SUSP	<p>Suspend.</p> <p>Host mode:</p> <ul style="list-style-type: none"> The port enabled bit (PE) and suspend (SUSP) bit define the port states as follows: <p>0x Disable 10 Enable 11 Suspend</p> <ul style="list-style-type: none"> 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 clears the FPR bit. A write of zero to this bit is ignored by the host controller. If host software sets this bit to a one when the port is not enabled (that is, port enabled bit is a zero) the results are undefined. This field is zero if Port Power (PP) is zero in host mode. <p>Device mode:</p> <p>1 Port in suspend state. 0 Port not in suspend state. Default. In device mode this bit is a read-only status bit.</p>

Table 16-22. PORTSC Register Field Descriptions (continued)

Bits	Name	Description
6	FPR	<p>Force port resume. This bit is not-EHCI compatible.</p> <p>1 Resume detected/driven on port. 0 No resume (K-state) detected/driven on port.</p> <p>Host mode:</p> <ul style="list-style-type: none"> • 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. 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. <p>Device mode:</p> <ul style="list-style-type: none"> • After the device has been in Suspend State for 5 msec or more, software must set this bit to one to drive resume signaling before clearing. The USB DR controller will set this bit to one if a J-to-K transition is detected while the port is in the Suspend state. The bit is cleared when the device returns to normal operation.
5	OCC	<p>Over-current change. This bit gets set when there is a change to over-current active. Software clears this bit by writing a one to this bit position.</p> <p>Host/OTG mode:</p> <ul style="list-style-type: none"> • The user can provide over-current detection to the USB_n_PWRFAULT signal for this condition. <p>Device mode:</p> <ul style="list-style-type: none"> • This bit must always be 0. <p>1 Over current detect. 0 No over current.</p>
4	OCA	<p>Over-current active. This bit will automatically transition from one to zero when the over current condition is removed.</p> <p>Host/OTG mode:</p> <ul style="list-style-type: none"> • The user can provide over-current detection to the USB_n_PWRFAULT signal for this condition. <p>Device mode:</p> <ul style="list-style-type: none"> • This bit must always be 0. <p>1 Port currently in over-current condition. 0 Port not in over-current condition.</p>
3	PEC	<p>Port enable/disable change.</p> <p>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.[</p> <p>In device mode:</p> <ul style="list-style-type: none"> • The device port is always enabled. (This bit is zero.) <p>1 Port disabled. 0 No change.</p> <p>This field is zero if Port Power(PP) is zero.</p>

Table 16-22. PORTSC Register Field Descriptions (continued)

Bits	Name	Description
2	PE	<p>Port enabled/disabled.</p> <p>Host mode:</p> <ul style="list-style-type: none"> 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. <p>Device mode:</p> <ul style="list-style-type: none"> The device port is always enabled. (This bit is one.)
1	CSC	<p>Connect change status.</p> <p>Host mode:</p> <ul style="list-style-type: none"> This bit indicates a change has occurred in the port's Current Connect Status. the controller sets this bit for all changes to the port device connect status, even if system software has not cleared an existing connect status change. For example, the insertion status changes twice before system software has cleared the changed condition, hub hardware is 'setting' an already-set bit (i.e., the bit will remain set). Software clears this bit by writing a one to it. <p>1 Connect Status has changed. 0 No change.</p> <ul style="list-style-type: none"> This field is zero if Port Power(PP) is zero. <p>Device mode:</p> <ul style="list-style-type: none"> This bit is undefined.
0	CCS	<p>Current connect status.</p> <p>Host mode:</p> <p>1 Device is present 0 No device present.</p> <p>This field is zero if Port Power(PP) is zero in host mode.</p> <p>In device mode:</p> <p>1 Attached 0 Not attached.</p> <p>A one indicates that the device successfully attached and is operating in either high-speed or full-speed as indicated by the Port Speed (PSPD) bit in this register. A zero indicates that the device did not attach successfully or was forcibly disconnected by the software writing a zero to USBCMD[RS] (run bit). It does not state the device being disconnected or suspended.</p>

16.3.2.15 On-The-Go Status and Control (OTGSC)—Non-EHCI

This register is not defined in the EHCI specification. The USB DR module implements one On-The-Go (OTG) status and control register corresponding to Port 0.

The OTGSC register has four sections:

- OTG interrupt enables (Read/Write)
- OTG interrupt status (Read/Write to Clear)
- OTG status inputs (Read Only)
- OTG controls (Read/Write)

The status inputs are de-bounced using a 1-msec time constant. Values on the status inputs that do not persist for more than 1 msec will not cause an update of the status inputs, or cause and OTG interrupt.

Offset 0x1A4

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	1	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	—	DPS	1msT	BSE	BSV	ASV	AVV	ID	—			DP	OT	—	VC	VD
W																
Reset	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0

Figure 16-21. OTG Status Control (OTGSC)

Table 16-23. 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. Set when data bus pulsing occurs on DP or DM. Data bus pulsing is only detected when USBMODE[CM] = Host (11) and PORTSC[PP] (port power) = Off (0). Software must write a one to clear this bit.
21	1msS	1-millisecond timer interrupt status. Set once every millisecond. Software must write a one to clear this bit.
20	BSEIS	B session end interrupt status. Set when VBus has fallen below the B session end threshold. Software must write a one to clear this bit.

Table 16-23. OTGSC Register Field Descriptions (continued)

Bits	Name	Description
19	BSVIS	B session valid interrupt status. Set when VBus has either risen above or fallen below the B session valid threshold (0.8 VDC). Software must write a one to clear this bit.
18	ASVIS	A session valid interrupt status. Set when VBus has either risen above or fallen below the A session valid threshold (0.8 VDC). Software must write a one to clear this bit.
17	AVVIS	A VBus valid interrupt status. Set when VBus has either risen above or fallen below the VBus valid threshold (4.4 VDC) on an A device. Software must write a one to clear this bit.
16	IDIS	USB ID interrupt status. Set when a change on the ID input has been detected. Software must write a one to clear this bit.
15	—	Reserved, should be cleared.
14	DPS	Data bus pulsing status 1 Pulsing detected on port 0 No pulsing on port
13	1msT	1 millisecond timer toggle. This bit toggles once per millisecond.
12	BSE	B session end 1 VBus is below the B session end threshold. 0 VBus is above the B session end threshold.
11	BSV	B session valid 1 VBus is above the B session valid threshold. 0 VBus is below the B session valid threshold.
10	ASV	A session valid 1 VBus is above the A session valid threshold. 0 VBus is below the A session valid threshold.
9	AVV	A VBus valid 1 VBus is above the A VBus valid threshold. 0 VBus is below the A VBus valid threshold.
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.

16.3.2.16 USB Mode Register (USBMODE)—Non-EHCI

The USB mode register, shown in Figure 16-22, is not defined in the EHCI specification. This register controls the operating mode of the module.



Figure 16-22. USB Mode (USBMODE)

Table 16-24 describes the USB mode register fields.

Table 16-24. USBMODE Register Field Descriptions

Bits	Name	Description
31–5	—	Reserved, should be cleared.
4	SDIS	Stream disable In host mode, setting this bit ensures that overruns/underruns of the latency FIFO are eliminated for low bandwidth systems where the RX and TX buffers are sufficient to contain the entire packet. Enabling stream disable also has the effect of ensuring the TX latency is filled to capacity or a complete packet has been stored 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, setting this bit disables double priming on both RX and TX for low bandwidth systems. This mode ensures that when the RX and TX buffers are sufficient to contain an entire packet that the standard double buffering scheme is disabled to prevent overruns/underruns in bandwidth limited systems. Note that in high-speed mode, all packets received are responded to with a NYET handshake when stream disable is active.
3	SLOM	Setup lockout mode. In device mode, this bit controls behavior of the setup lock mechanism. See Section 16.8.3.5, “Control Endpoint Operation Model.” 1 Setup lockouts off. DCD requires use of setup data buffer tripwire in USBCMD (SUTW). 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 USBCMD[RST] before reprogramming this register. 00 Idle. 01 Reserved, should be cleared. 10 Device controller. 11 Host controller. Defaults to the idle state and needs to be initialized to the desired operating mode after reset.

16.3.2.17 Endpoint Setup Status Register (ENDPTSETUPSTAT)—Non-EHCI

The endpoint setup status register, shown in [Figure 16-23](#), is not defined in the EHCI specification. This register contains the endpoint setup status. It is only used in device mode.

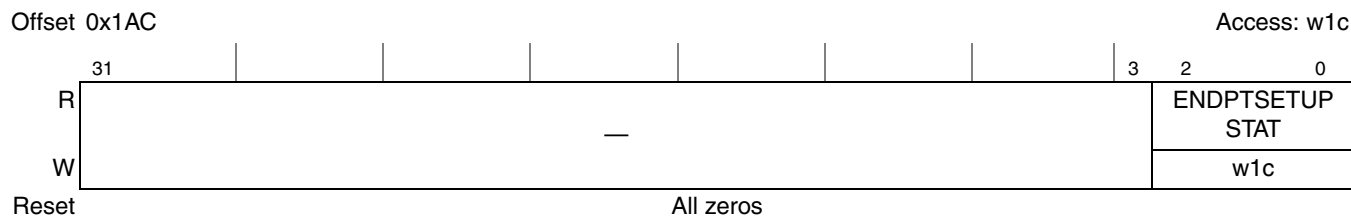


Figure 16-23. Endpoint Setup Status (ENDPTSETUPSTAT)

[Table 16-25](#) describes the endpoint setup status register fields.

Table 16-25. ENDPTSETUPSTAT Register Field Descriptions

Bits	Name	Description
31–3	—	Reserved, should be cleared.
2–0	ENDPTSETUP STAT	Setup endpoint status. For every setup transaction that is received, a corresponding bit in this register is set. Software must clear or acknowledge the setup transfer by writing a one to a respective bit after it has read the setup data from queue head. The response to a setup packet as in the order of operations and total response time is crucial to limit bus time outs while the setup lockout mechanism is engaged. This register is only used in device mode.

16.3.2.18 Endpoint Initialization Register (ENDPTPRIME)—Non-EHCI

The endpoint initialization register, shown in [Figure 16-23](#), is not defined in the EHCI specification. This register is used to initialize endpoints. It is only used in device mode.

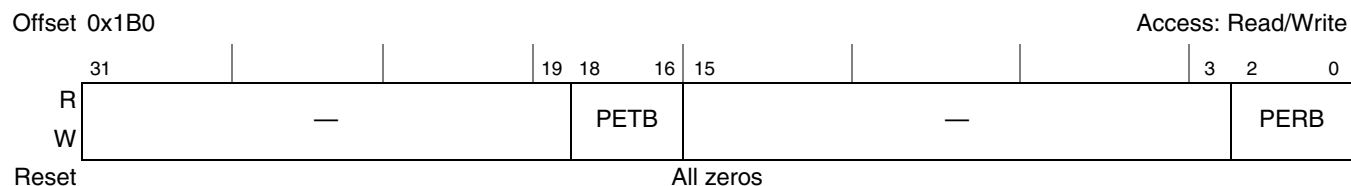


Figure 16-24. Endpoint Initialization (ENDPTPRIME)

Table 16-26 describes the endpoint initialization register fields.

Table 16-26. ENDPTPRIME Register Field Descriptions

Bits	Name	Description
31–19	—	Reserved, should be cleared.
18–16	PETB	Prime endpoint transmit buffer. For each endpoint a corresponding bit is used to request that a buffer prepared for a transmit operation in order to respond to a USB IN/INTERRUPT transaction. Software should write a one to the corresponding bit when posting a new transfer descriptor to an endpoint. Hardware will automatically use this bit to begin parsing for a new transfer descriptor from the queue head and prepare a transmit buffer. Hardware will clear this bit when the associated endpoint(s) is (are) successfully primed. PETB[2] (bit 18 of the register) corresponds to endpoint 2. Note that these bits are momentarily set by hardware during hardware re-priming operations when a dTD is retired, and the dQH is updated.
15–3	—	Reserved, should be cleared.
2–0	PERB	Prime endpoint receive buffer. For each endpoint, a corresponding bit is used to request a buffer prepare for a receive operation in order to respond to a USB OUT transaction. Software should write a one to the corresponding bit whenever posting a new transfer descriptor to an endpoint. Hardware will automatically use this bit to begin parsing for a new transfer descriptor from the queue head and prepare a receive buffer. Hardware will clear this bit when the associated endpoint(s) is (are) successfully primed. PERB[2] corresponds to endpoint 2. Note that these bits are momentarily set by hardware during hardware re-priming operations when a dTD is retired, and the dQH is updated.

16.3.2.19 Endpoint Flush Register (ENDPTFLUSH)—Non-EHCI

The endpoint flush register, shown in Figure 16-25, is not defined in the EHCI specification. This register is only used in device mode.

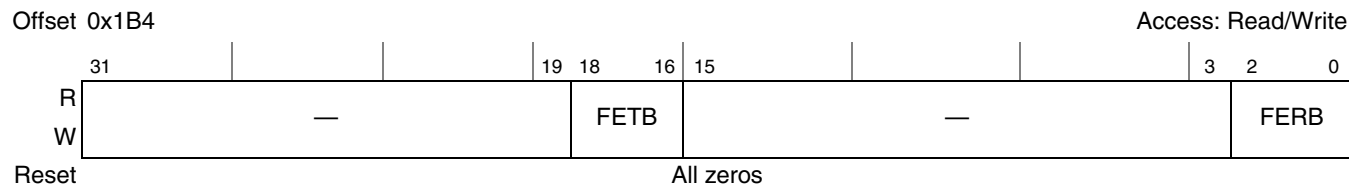


Figure 16-25. Endpoint Flush (ENDPTFLUSH)

Table 16-27 describes the endpoint flush register fields.

Table 16-27. ENDPTFLUSH Register Field Descriptions

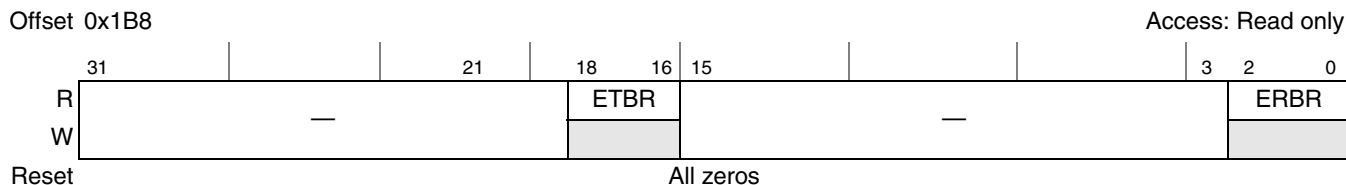
Bits	Name	Description
31–19	—	Reserved, should be cleared.
18–16	FETB	Flush endpoint transmit buffer. Writing a one to a bit(s) in this register will cause the associated endpoint(s) to clear any primed buffers. If a packet is in progress for one of the associated endpoints, then that transfer will continue until completion. Hardware will clear this register after the endpoint flush operation is successful. FETB[2] (bit 18 of the register) corresponds to endpoint 2.

Table 16-27. ENDPTFLUSH Register Field Descriptions (continued)

Bits	Name	Description
15–3	—	Reserved, should be cleared.
2–0	FERB	Flush endpoint receive buffer. Writing a one to a bit(s) will cause the associated endpoint(s) to clear any primed buffers. If a packet is in progress for one of the associated endpoints, then that transfer will continue until completion. Hardware will clear this register after the endpoint flush operation is successful. FERB[2] corresponds to endpoint 2.

16.3.2.20 Endpoint Status Register (ENDPTSTATUS)—Non-EHCI

The endpoint status register, shown in [Figure 16-25](#), is not defined in the EHCI specification. This register is only used in device mode.


Figure 16-26. Endpoint Status (ENDPTSTATUS)

[Table 16-28](#) describes the endpoint status fields.

Table 16-28. ENDPTSTATUS Register Field Descriptions

Bits	Name	Description
31–19	—	Reserved, should be cleared
18–16	ETBR	Endpoint transmit buffer ready. One bit for each endpoint indicates status of the respective endpoint buffer. This bit is set by the hardware as a response to receiving a command from a corresponding bit in the ENDPTPRIME register. There will always be a delay between setting a bit in the ENDPTPRIME register and endpoint indicating ready. This delay time varies based upon the current USB traffic and the number of bits set in the ENDPTPRIME register. Buffer ready is cleared by USB reset, by the USB DMA system, or through the ENDPTFLUSH register. ETBR[2] (bit 18 of the register) corresponds to endpoint 2. Note that these bits are momentarily cleared by hardware during hardware endpoint re-priming operations when a dTD is retired, and the dQH is updated.
15–3	—	Reserved, should be cleared
2–0	ERBR	Endpoint receive buffer ready. One bit for each endpoint indicates status of the respective endpoint buffer. This bit is set by the hardware as a response to receiving a command from a corresponding bit in the ENDPTPRIME register. There will always be a delay between setting a bit in the ENDPTPRIME register and endpoint indicating ready. This delay time varies based upon the current USB traffic and the number of bits set in the ENDPTPRIME register. Buffer ready is cleared by USB reset, by the USB DMA system, or through the ENDPTFLUSH register. ERBR[2] corresponds to endpoint 2. Note that these bits are momentarily cleared by hardware during hardware endpoint re-priming operations when a dTD is retired, and the dQH is updated.

16.3.2.21 Endpoint Complete Register (ENDPTCOMPLETE)—Non-EHCI

The endpoint complete register, shown in [Figure 16-25](#), is not defined in the EHCI specification. This register is only used in device mode.

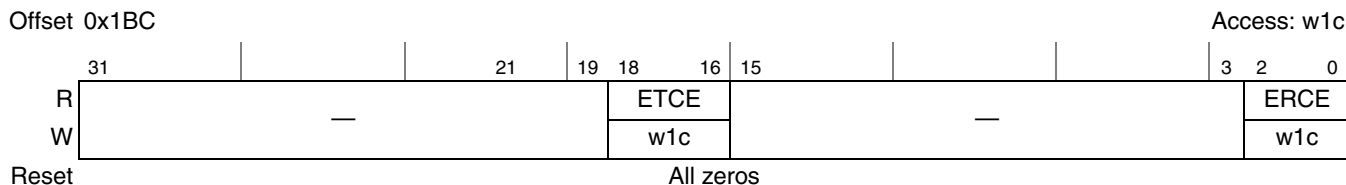


Figure 16-27. Endpoint Complete (ENDPTCOMPLETE)

Table 16-29 describes the endpoint complete register fields.

Table 16-29. ENDPTCOMPLETE Register Field Descriptions

Bits	Name	Description
31–19	—	Reserved, should be cleared
18–16	ETCE	Endpoint transmit complete event. Each bit indicates a transmit event (IN/INTERRUPT) occurred and software should read the corresponding endpoint queue to determine the endpoint status. If the corresponding IOC bit is set in the Transfer Descriptor, then this bit is set simultaneously with the USBINT. Writing a one will clear the corresponding bit in this register. ETCE[2] (bit 18 of the register) corresponds to endpoint 2.
15–3	—	Reserved, should be cleared
2–0	ERCE	Endpoint receive complete event. Each bit indicates a received event (OUT/SETUP) occurred and software should read the corresponding endpoint queue to determine the transfer status. If the corresponding IOC bit is set in the Transfer Descriptor, then this bit is set simultaneously with the USBINT. Writing a one will clear the corresponding bit in this register. ERCE[2] corresponds to endpoint 2.

16.3.2.22 Endpoint Control Register 0 (ENDPTCTRL0)—Non-EHCI

Endpoint control register 0, shown in Figure 16-25, is not defined in the EHCI specification. Every device will implement endpoint 0 as a control endpoint.

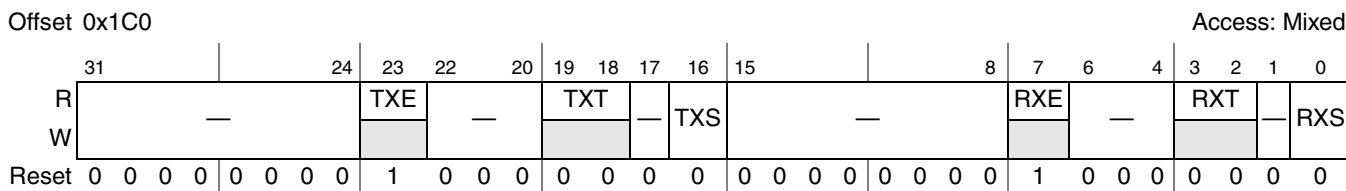


Figure 16-28. Endpoint Control 0 (ENDPTCTRL0)

Table 16-30 describes the endpoint control register 0 fields.

Table 16-30. ENDPTCTRL0 Register Field Descriptions

Bits	Name	Description
31–24	—	Reserved, should be cleared.
23	TXE	TX endpoint enable. Endpoint zero is always enabled. 0 Disable 1 Enable
22–20	—	Reserved, should be cleared.
19–18	TXT	TX endpoint type. Endpoint zero is always a control endpoint (00).

Table 16-30. ENDPTCTRL0 Register Field Descriptions (continued)

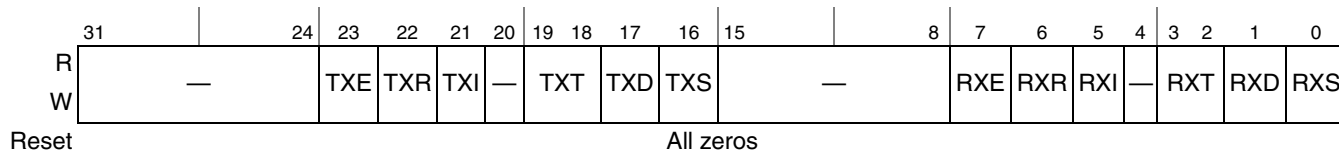
Bits	Name	Description
17	—	Reserved, should be cleared.
16	TXS	TX endpoint stall. Software can write a one to this bit to force the endpoint to return a STALL handshake to the Host. It will continue returning STALL until the bit is cleared by software or it will automatically be cleared upon receipt of a new SETUP request. 1 Endpoint stalled 0 Endpoint OK
15–8	—	Reserved, should be cleared.
7	RXE	RX endpoint enable. Endpoint zero is always enabled. 0 Disabled 1 Enabled
6–4	—	Reserved, should be cleared.
3–2	RXT	RX endpoint type. Endpoint zero is always a control endpoint (00).
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

16.3.2.23 Endpoint Control Register n (ENDPTCTRL n)—Non-EHCI

The endpoint control n registers, shown in [Figure 16-29](#), are not defined in the EHCI specification. There is an ENDPTCTRL n register of each endpoint in a device.

Offset 0x1C4 (ENDPTCTRL1), 0x1C8 (ENDPTCTRL2),

Access: Read/Write


Figure 16-29. Endpoint Control 1 to 5 (ENDPTCTRL n)

[Table 16-31](#) describes the endpoint control n register fields.

Table 16-31. ENDPTCTRL n Register Field Descriptions

Bits	Name	Description
31–24	—	Reserved, should be cleared
23	TXE	TX endpoint enable 0 Disabled 1 Enabled
22	TXR	TX data toggle reset. Whenever a configuration event is received for this endpoint, software must write a one to this bit in order to synchronize the data PID's between the Host and device.

Table 16-31. ENDPCTRL n Register Field Descriptions (continued)

Bits	Name	Description
21	TXI	TX data toggle inhibit. Used only for test and should always be written as zero. Writing a one to this bit will cause this endpoint to ignore the data toggle sequence and always transmit DATA0 for a data packet. 0 PID sequencing enabled 1 PID sequencing disabled
20	—	Reserved, should be cleared
19–18	TXT	TX endpoint type 00 Control 01 Isochronous 10 Bulk 11 Interrupt Note: When only one endpoint (RX or TX, but not both) of an endpoint pair is used, the unused endpoint should be configured as a bulk type endpoint.
17	TXD	TX endpoint data source. This bit should always be written as 0, which selects the dual port memory/DMA engine as the source.
16	TXS	TX endpoint stall. This bit is set automatically upon receipt of a SETUP request if this endpoint is not configured as a control endpoint. It is cleared automatically upon receipt of a SETUP request if this endpoint is configured as a control endpoint. Software can write a one to this bit to force the endpoint to return a STALL handshake to the host. It will continue to returning STALL until this bit is either cleared by software or automatically cleared as above. 0 Endpoint OK 1 Endpoint stalled
15–8	—	Reserved, should be cleared
7	RXE	RX endpoint enable 0 Disabled 1 Enabled
6	RXR	RX data toggle reset. Whenever a configuration event is received for this endpoint, software must write a one to this bit in order to synchronize the data PID's between the Host and device.
5	RXI	RX data toggle inhibit. This bit is only used for test and should always be written as zero. Writing a one to this bit will cause this endpoint to ignore the data toggle sequence and always accept data packets regardless of their data PID. 1 PID sequencing enabled 0 PID sequencing disabled
4	—	Reserved, should be cleared
3–2	RXT	RX endpoint type 00 Control 01 Isochronous 10 Bulk 11 Interrupt 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.

Table 16-31. ENDPTCTRL n Register Field Descriptions (continued)

Bits	Name	Description
1	RXD	RX endpoint data sink. This bit should always be written as 0, which selects the dual port memory/DMA engine as the sink.
0	RXS	RX endpoint stall. This bit is set automatically upon receipt of a SETUP request if this endpoint is not configured as a control endpoint. It is cleared automatically upon receipt a SETUP request if this endpoint is configured as a control endpoint, Software can write a one to this bit to force the endpoint to return a STALL handshake to the host. It will continue to returning STALL until this bit is either cleared by software or automatically cleared as above, 1 Endpoint stalled 0 Endpoint OK

16.3.2.24 SNOOP1 and SNOOP2—Non-EHCI

Figure 16-30 shows the SNOOP1 and SNOOP2 registers. Note that these registers use big-endian byte ordering and are not defined in the EHCI specification. The SNOOP1 and SNOOP2 registers provide snooping control and address range selection function. Transactions that hit a snooping window will generate cache coherent transactions on the internal CSB bus. When the five lower bits (SNOOP 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 16-30. Snoop 1 and Snoop 2 (SNOOP n)

Table 16-32 describes the SNOOP n register fields.

Table 16-32. SNOOP n Register Field Descriptions

Bits	Name	Description
0–19	Snoop address	The starting base address for which transactions are snooped.

Table 16-32. SNOOP_n Register Field Descriptions (continued)

Bits	Name	Description
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]

16.3.2.25 Age Count Threshold Register (AGE_CNT_THRESH)—Non-EHCI

Note that this register uses big-endian byte ordering and is not defined in the EHCI specification. The age count threshold (AGE_CNT_THRESH) register provides the aging counter threshold value used to determine the priority state of the USB DR controller’s internal system interface. This is used to increase the priority state of the module’s system interface from zero to one. The actual priority level on the system bus for each state is defined by the PRI_CTRL register. See [Section 7.3.1.1, “Address Bus Arbitration with PRIORITY\[0:1\],”](#) for more details on bus priority. The threshold value is in units of *csb_clk* cycles. This register should be written during system initialization or during normal system operation when the system bus interface is idle. It can be read at any time.

If the aging counter is less than the AGE_CNT_THRESH value, priority state zero is chosen. If the aging counter is greater than or equal to the AGE_CNT_THRESH value, priority state one is chosen.

The aging counter begins to count from zero when a bus access is requested. It increments every bus cycle until the bus transaction completes. At the completion of a bus transaction, the counter is synchronously reset to zero. If there are any outstanding bus requests, the aging counter will then begin counting immediately.

The AGE_CNT_THRESH is compared against the value of the aging counter during each clock cycle of the current transaction. If AGE_CNT_THRESH is equal to zero, priority state one is always chosen. If the aging counter is less than the AGE_CNT_THRESH value, priority state zero is selected. If the aging counter is greater than or equal to the AGE_CNT_THRESH value, priority state one is selected.

The two priority states of the aging counter function each have corresponding register bits which are programmed by the CPU. Thus, when the aging counter function is at priority state zero, `PRI_CTRL[30–31]` are selected and used to drive bus priority levels. When the aging counter function is at priority state one, `PRI_CTRL[28–29]` are selected and used to drive the priority.

Figure 16-31 shows the age count threshold register.

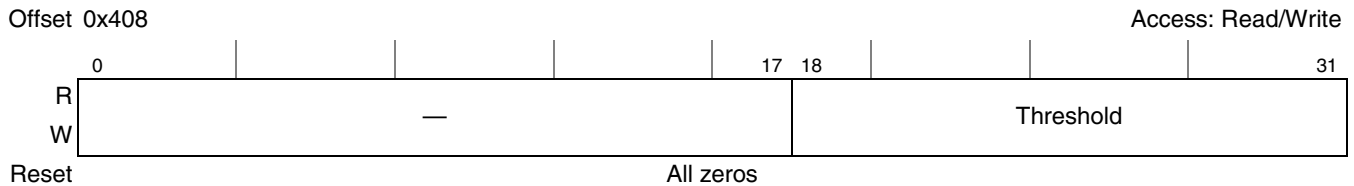


Figure 16-31. Age Count Threshold (AGE_CNT_THRESH)

Table 16-33 describes the age count threshold register fields.

Table 16-33. AGE_CNT_THRESH Register Field Descriptions

Bits	Name	Description
0–17	—	Reserved, should be cleared
18–31	Threshold	Aging counter threshold value.

The setting of `AGE_CNT_THRESH` is highly dependent on both the mix of other controllers operating on the system bus as well as the kind of traffic moving through the USB controller. A recommended approach is first to try leaving the aging mechanism disabled and see if the USB meets performance requirements. If USB performance does not meet application requirements, try the following settings:

- Set `PRI_CTRL[pri_lv10]` to 0.
- Set `tPRI_CTRL[pri_lv11]` to 3.
- Set `AGE_CNT_THRESH` to 40.

This combination works for a wide variety of applications. If this combination still does not meet application requirements, try lowering `AGE_CNT_THRESH` by 5. On the contrary, the setting 40 may be too conservative for some applications. If USB performance is acceptable at 40, try raising the value in increments of 5. Raising `AGE_CNT_THRESH` benefits the other controllers on the system bus by reducing the frequency that this USB controller raises its priority to the arbiter.

16.3.2.26 Priority Control Register (PRI_CTRL)—Non-EHCI

Note that this register uses big-endian byte ordering and is not defined in the EHCI specification. The priority control (`PRI_CTRL`) register sets the priority level for each of two priority states. The priority state is determined by the value programmed in the `AGE_CNT_THRESH` register and the number of `csb_clk` cycles that a particular transaction takes to complete.

Figure 16-32 shows the priority control register.

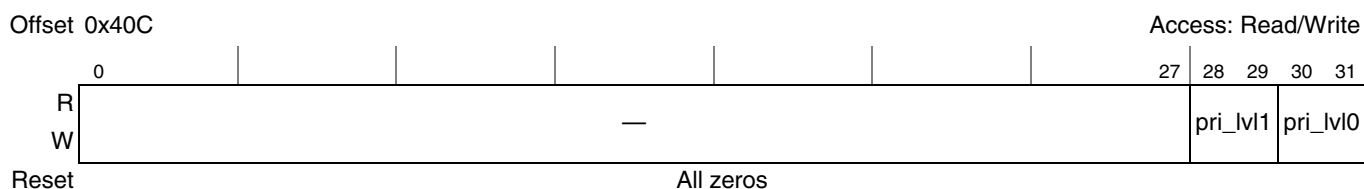


Figure 16-32. Priority Control (PRI_CTRL)

Table 16-34 describes the priority control register fields.

Table 16-34. PRI_CTRL Register Field Descriptions

Bits	Name	Description
0–27	—	Reserved, should be cleared
28–29	pri_lvl1	Priority level for priority state 1 (11 being the highest priority and 00 the lowest)
30–31	pri_lvl0	Priority level for priority state 0 (11 being the highest priority and 00 the lowest)

16.3.2.27 System Interface Control Register (SI_CTRL)—Non-EHCI

Note that this register uses big-endian byte ordering and is not defined in the EHCI specification. The system interface control register (SI_CTRL) controls various functions pertaining to the internal system interface.

Figure 16-33 shows the system interface control register.

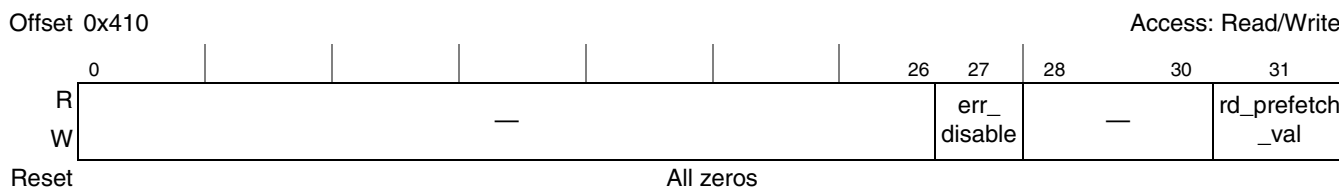


Figure 16-33. System Interface Control Register (SI_CTRL)

Table 16-35 describes the system interface control register fields.

Table 16-35. SI_CTRL Register Field Descriptions

Bits	Name	Description
0–26	—	Reserved, should be cleared
27	err_disable	When this bit is set, it causes the controller to ignore system bus errors. If cleared the controller responds according to the values set in USBSTS[SEI] and USBINT[SEE]. 0 enable 1 disable

Table 16-35. SI_CTRL Register Field Descriptions (continued)

Bits	Name	Description
28–30	—	Reserved, should be cleared
31	rd_prefetch_val	<p>Selects whether 32 bytes or 64 bytes are fetched during burst read transactions at the system interface. When this bit is zero, 64 bytes are fetched and when it is one, 32 bytes are fetched. The setting of rd_prefetch_val must match the setting of the larger of TXPBURST and RXPBURST fields in the BURSTSIZE register. If either of these fields is 64 bytes, then rd_prefetch_val must be left cleared. Otherwise, this value should be set.</p> <p>0 64-byte fetch 1 32-byte fetch</p>

16.3.2.28 USB General Purpose Register (CONTROL)—Non-EHCI

Note that this register uses big-endian byte ordering and is not defined in the EHCI specification. The USB general purpose (CONTROL) register contains the general-purpose IP control register outputs and is shown in [Figure 16-34](#).

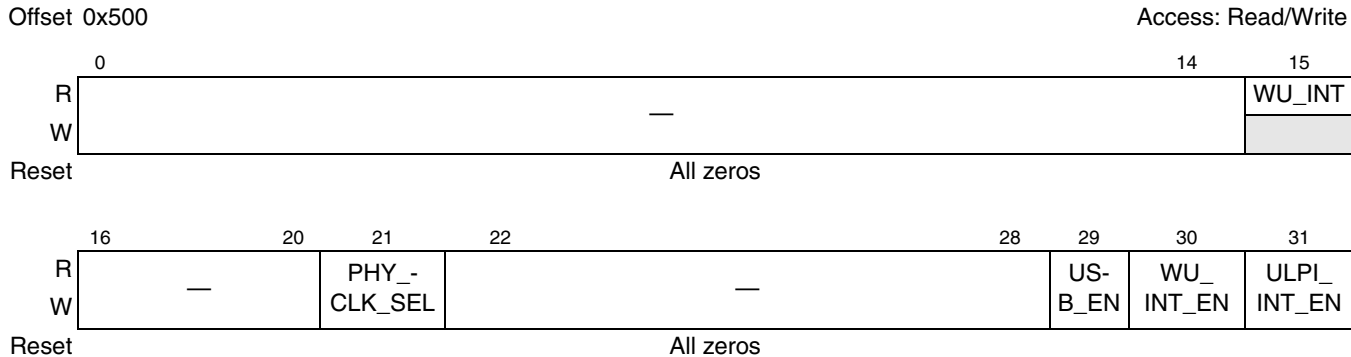


Figure 16-34. USB General-Purpose Register (CONTROL)

[Table 16-36](#) describes the USB general-purpose register fields.

Table 16-36. CONTROL Field Descriptions

Bits	Name	Description
0–14	—	Reserved
15	WU_INT	<p>Reflects the state of the wake up interrupt. The wake up interrupt signal is asserted when a wake-up event occurs while in a low-power suspend state. If WU_INT_EN is set, this WU_INT signal generates an interrupt to the system to indicate wake up servicing is required. WU_INT will remain set until the USB controller is exited from the low power by clearing the PORTSC[PHCD] bit.</p> <p>0 Normal operation or Low Power mode waiting for wakeup event 1 Low power wakeup event has occurred</p>
16–20	—	Reserved
21	PHY_CLK_SEL	<p>Select the source of the USB link controller transceiver clock. When cleared, the UTMI PHY is the source of the clock. When set, the clock is sourced from the external ULPI PHY.</p> <p>0 UTMI is clock source 1 ULPI is clock source</p>

Table 16-36. CONTROL Field Descriptions (continued)

Bits	Name	Description
22–28	—	Reserved
29	USB_EN	1 Enable 0 Disable ULPI mode: In safe mode, all USB interface signals are put into input mode or driven inactive, except for SUSPEND_STP which is driven high. Also, the input signal DIR is forced to appear high to the controller. This prevents any start-up problems that otherwise could occur if the PHY and the controller take significantly different times to complete power-on reset. 1 Normal operation 0 Safe mode
30	WU_INT_EN	This bit is used to mask/unmask the system wakeup interrupt signal 0 System wakeup interrupt disabled 1 System wakeup interrupt enabled Note: PORTSC[PHCD] bit must be set for the system wakeup interrupt generation.
31	ULPI_INT_EN	Used to enable the ULPI low power wakeup interrupt from the PHY when the PHY is in low power mode only. 0 ULPI low power wakeup interrupt disabled 1 ULPI low power wakeup interrupt enabled Note: PORTSC[PHCD] bit must be set

16.4 Functional Description

The USB DR module can be broken down into functional sub-blocks, which are described below.

16.4.1 System Interface

The system interface block contains all the control and status registers that allow a processor to interface to the USB DR module. These registers allow the processor to control the configuration of the module, ascertain the capabilities of the module, and control the module's operation. It also has registers to control snoopability and priority of the DMA interface.

16.4.2 DMA Engine

The module contains a local DMA engine. The DMA engine interfaces internally to the CSB. It is responsible for moving all of the data to be transferred over the USB between the module and buffers in system memory. Like the system interface block, the DMA engine block uses a simple synchronous bus signaling protocol that eases connections to a number of different standard buses.

The DMA controller must access both control information and packet data from system memory. The control information is contained in link list-based queue structures. The DMA controller has state machines that are able to parse data structures defined in the EHCI specification. In host mode, the data structures are EHCI compliant and represent queues of transfers to be performed by the host controller, including the split-transaction requests that allow an EHCI controller to direct packets to FS and LS devices. In device mode, the data structures are designed to be similar to those in the EHCI specification and are used to allow device responses to be queued for each of the active pipes in the device.

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, multiple FIFO channels are maintained for each of the active endpoints in the system.

In host mode, the USB DR module uses a 512-byte Tx buffer and a 512-byte Rx buffer. Device operation uses a single 512-byte Rx buffer and a 512-byte Tx buffer for each endpoint. The 512-byte buffers allow the module to buffer a complete HS bulk packet.

16.4.4 PHY Interface

The USB DR module interfaces to any ULPI-compatible PHY. The primary function of the port controller block is to isolate the rest of the module from the transceiver, and to move all of the transceiver signaling into the primary clock domain of the module. This allows the module to run synchronously with the system processor and its associated resources.

Due to pin count limitations the module only supports certain combinations of PHY interfaces and USB functionality. Refer to [Table 16-37](#) for more information.

Table 16-37. Supported PHY Interfaces

PHY	Function
ULPI	Host/Device/OTG

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 with queue heads and queue element transfer descriptors. These data structures are optimized to reduce the total memory footprint of the schedule and to reduce (on average) the number of memory accesses needed to execute a USB transaction.

Note that software must ensure that no interface data structure reachable by the EHCI host controller spans a 4K-page boundary.

The data structures defined in this section are (from the host controller’s perspective) a mix of read-only and read/writable fields. The host controller will preserve the read-only fields on all data structure writes.

16.5.1 Periodic Frame List

Figure 16-35 shows the organization of the periodic schedule. This schedule is for all periodic transfers (isochronous and interrupt). The periodic schedule is referenced from the operational registers space using the PERIODICLISTBASE address register and the FRINDEX register. The periodic schedule is based on an array of pointers called the periodic frame list. The PERIODICLISTBASE address register is combined with the FRINDEX register to produce a memory pointer into the frame list. The periodic frame list implements a sliding window of work over time.

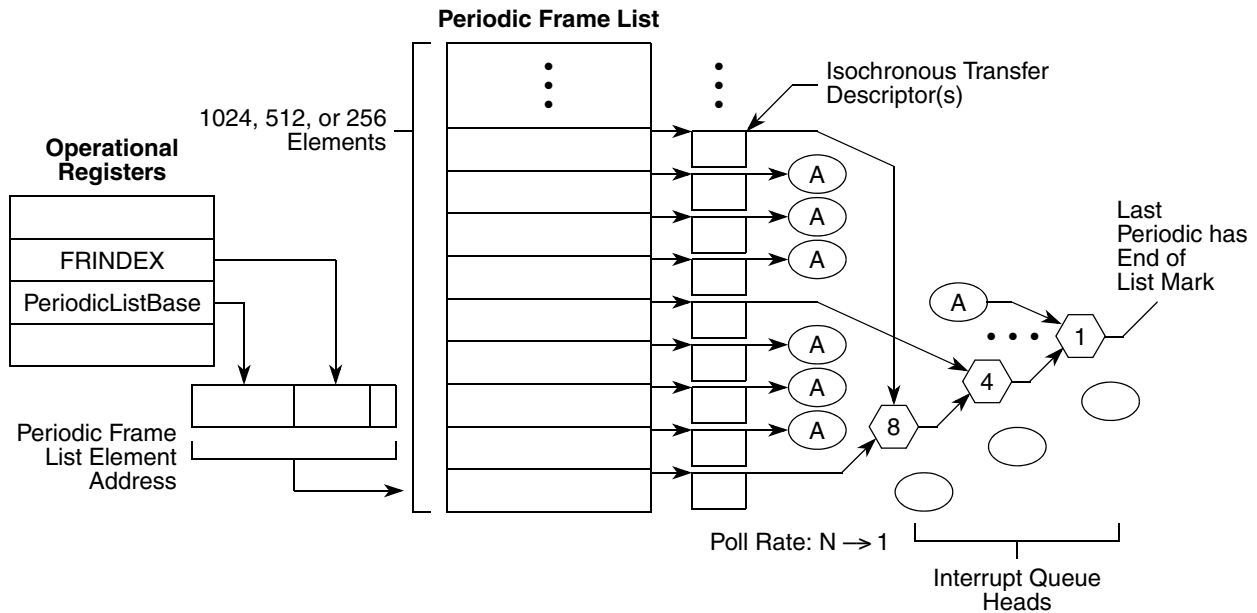


Figure 16-35. Periodic Schedule Organization

The periodic frame list is a 4K-page aligned array of Frame List Link pointers. The length of the frame list is programmable. The programmability of the periodic frame list is exported to system software through the HCCPARAMS register. The length can be selected by system software as one of 8, 16, 32, 64, 128, 256, 512 or 1024 elements. Programming the size (that is, the number of elements) is accomplished by system software writing the appropriate value into Frame List Size field in the USBCMD register.

Frame list link pointers direct the host controller to the first work item in the frame’s periodic schedule for the current microframe. The link pointers are aligned on DWord boundaries within the frame list.

Figure 16-36 shows the format for the frame list link pointer.

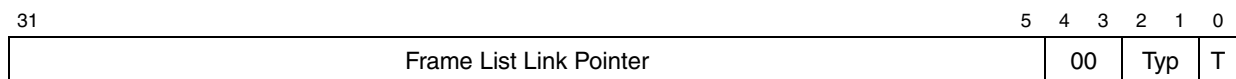


Figure 16-36. Frame List Link Pointer Format

Frame list link pointers always reference memory objects that are 32-byte aligned. The referenced object may be an isochronous transfer descriptor for high-speed devices, a split-transaction isochronous transfer

descriptor (for full-speed isochronous endpoints), or a queue head (used to support high-, full- and low-speed interrupt). System software should not place non-periodic schedule items into the periodic schedule. The least-significant bits in a frame list pointer are used to key the host controller in as to the type of object the pointer is referencing.

The least-significant bit is the T bit (bit 0). When this bit is set, the host controller never uses the value of the frame list pointer as a physical memory pointer. The Typ field indicates the exact type of data structure being referenced by this pointer. The value encodings for the Typ field are given in [Table 16-38](#).

Table 16-38. 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 16-37](#) shows the asynchronous schedule organization.

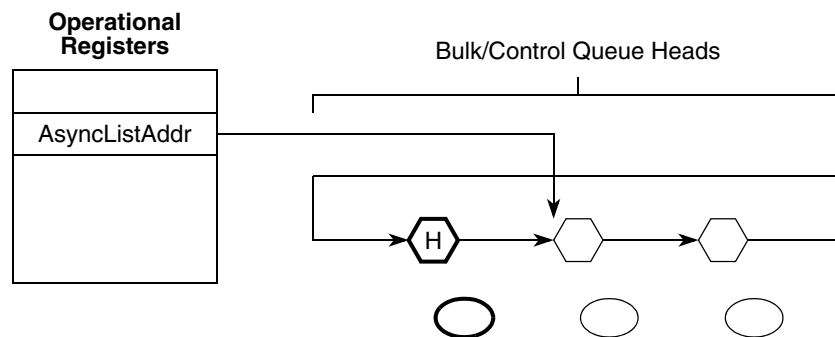


Figure 16-37. Asynchronous Schedule Organization

The asynchronous list is a simple circular list of queue heads. The ASYNCLISTADDR register is simply a pointer to the next queue head. This implements a pure round-robin service for all queue heads linked into the asynchronous list.

16.5.3 Isochronous (High-Speed) Transfer Descriptor (iTd)

[Figure 16-38](#) illustrates the format of an isochronous transfer descriptor. This structure is used only for high-speed isochronous endpoints. All other transfer types should use queue structures. Isochronous TDs must be aligned on a 32-byte boundary.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	offset
Next Link Pointer																											00	Typ	T	0x00		
Status 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 16-38. 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, as shown in [Table 16-39](#).

Table 16-39. Next Schedule Element Pointer

Bits	Name	Description
31–5	Link Pointer	Correspond to memory address signals [31:5], respectively. This field points to another isochronous transaction descriptor (iTD/siTD) or queue head (QH).
4–3	—	Reserved, should be cleared. These bits are reserved and their value has no effect on operation. Software should initialize this field to zero.
2–1	Typ	Indicates to the host controller whether the item referenced is an iTD, siTD or a QH. This allows the host controller to perform the proper type of processing on the item after it is fetched. Value encodings are: 00 iTD (isochronous transfer descriptor) 01 QH (queue head) 10 siTD (split transaction isochronous transfer descriptor) 11 FSTN (frame span traversal node)
0	T	Terminate 1 Link Pointer field is not valid. 0 Link Pointer field is valid.

16.5.3.2 iTD Transaction Status and Control List

DWords 1–8 constitute eight slots of transaction control and status. Each transaction description includes the following:

- Status results field
- Transaction length (bytes to send for OUT transactions and bytes received for IN transactions).
- Buffer offset. The PG and Transaction n Offset fields are used with the buffer pointer list to construct the starting buffer address for the transaction.

The host controller uses the information in each transaction description, plus the endpoint information contained in the first three DWords of the buffer page pointer list, to execute a transaction on the USB.

Table 16-40 shows the iTD transaction status and control fields.

Table 16-40. iTD Transaction Status and Control

Bits	Name	Description
31–28	Status	Records the status of the transaction executed by the host controller for this slot. This field is a bit vector with the following encoding: 31 Active. Set by software to enable the execution of an isochronous transaction by the host controller. When the transaction associated with this descriptor is completed, the host controller clears this bit indicating that a transaction for this element should not be executed when it is next encountered in the schedule. 30 Data buffer error. Set by the host controller during status update to indicate that the host controller is unable to keep up with the reception of incoming data (overflow) or is unable to supply data fast enough during transmission (underflow). If an overflow 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) = 24 576 bytes to be moved with this data structure, regardless of the alignment offset of the first page.

Since each pointer is a 4 K-aligned page pointer, the least-significant 12 bits in several of the page pointers are used for other purposes.

Table 16-41–Table 16-44 describe buffer pointer page *n*.

Table 16-41. Buffer Pointer Page 0 (Plus)

Bits	Name	Description
31–12	Buffer Pointer (Page 0)	A 4K-aligned pointer to physical memory. Corresponds to memory address bits 31–12.
11–8	EndPt	Selects the particular endpoint number on the device serving as the data source or sink.
7	—	Reserved, should be cleared. Reserved for future use and should be initialized by software to zero.
6–0	Device Address	This field selects the specific device serving as the data source or sink.

Table 16-42. iTD Buffer Pointer Page 1 (Plus)

Bits	Name	Description
31–12	Buffer Pointer (Page 1)	This is a 4K-aligned pointer to physical memory. Corresponds to memory address bits 31–12.
11	I/O	Direction (I/O). This field encodes whether the high-speed transaction should use an IN or OUT PID. 0 OUT 1 IN
10–0	Maximum Packet Size	This directly corresponds to the maximum packet size of the associated endpoint (<i>wMaxPacketSize</i>). This field is used for high-bandwidth endpoints where more than one transaction is issued per transaction description (for example, per microframe). This field is used with the <i>Multi</i> field to support high-bandwidth pipes. This field is also used for all IN transfers to detect packet babble. Software should not set a value larger than 1024 (0x400). Any value larger yields undefined results.

Table 16-43. Buffer Pointer Page 2 (Plus)

Bits	Name	Description
31–12	Buffer Pointer (Page 2)	This is a 4K-aligned pointer to physical memory. Corresponds to memory address bits 31–12.
11–2	—	Reserved, should be cleared. This bit reserved for future use and should be cleared.
1–0	Mult	Indicates to the host controller the number of transactions that should be executed per transaction description (for example, per microframe). 00 Reserved, should be cleared. A zero in this field yields undefined results. 01 One transaction to be issued for this endpoint per microframe 10 Two transactions to be issued for this endpoint per microframe 11 Three transactions to be issued for this endpoint per microframe

Table 16-44. Buffer Pointer Page 3–6

Bits	Name	Description
31–12	Buffer Pointer	This is a 4K-aligned pointer to physical memory. Corresponds to memory address bits 31–12.
11–2	—	Reserved, should be cleared. These bits reserved for future use and should be cleared.

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 16-39 shows the split-transaction isochronous transfer descriptor (siTD).

31		30		29		28		27		26		25		24		23		22		21		20		19		18		17		16		15		14		13		12		11		10		9		8		7		6		5		4		3		2		1		0		offset
Next Link Pointer																												00	Typ	T	0x00																																	
I/O	Port Number				0	Hub Address				0000	EndPt	0	Device Address																0x04																																			
0000_0000_0000_00000												μFrame C-mask				μFrame S-mask																0x08																																
ioc	P ¹	0000				Total Bytes to Transfer ¹				μFrame C-prog-mask ¹				Status ¹																0x0C																																		
Buffer Pointer (Page 0)												Current Offset ¹																0x10																																				
Buffer Pointer (Page 1)												000_0000				TP ¹	T-count ¹																0x14																															
Back Pointer																												0000				T	0x18																															

Figure 16-39. 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 16-45 describes the next link pointer fields.

Table 16-45. Next Link Pointer

Bits	Name	Description
31–5	Next Link Pointer	This field contains the address of the next data object to be processed in the periodic list and corresponds to memory address signals [31:5], respectively.
4–3	—	Reserved, should be cleared. These bits must be written as zeros.
2–1	Typ	Indicates to the host controller whether the item referenced is an iTD/siTD or a QH. This allows the host controller to perform the proper type of processing on the item after it is fetched. Value encodings are: 00 iTD (isochronous transfer descriptor) 01 QH (queue head) 10 siTD (split transaction isochronous transfer descriptor) 11 FSTN (frame span traversal node)
0	T	Terminate. 0 Link pointer is valid. 1 Link pointer field is not valid.

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 microframe scheduling control.

Table 16-46 describes the endpoint and transaction translator characteristics.

Table 16-46. Endpoint and Transaction Translator Characteristics

Bits	Name	Description
31	I/O	Direction (I/O). This field encodes whether the full-speed transaction should be an IN or OUT. 0 OUT 1 IN
30–24	Port Number	This field is the port number of the recipient transaction translator.
23	—	Reserved, should be cleared. Bit reserved and should be cleared.
22–16	Hub Address	This field holds the device address of the companion controllers' hub.
15–12	—	Reserved, should be cleared. Field reserved and should be cleared.
11–8	EndPt	Endpoint Number. Selects the particular endpoint number on the device serving as the data source or sink.
7	—	Reserved, should be cleared. Bit is reserved for future use. It should be cleared.
6–0	Device Address	Selects the specific device serving as the data source or sink.

Table 16-40 describes the microframe schedule control.

Table 16-47. Microframe Schedule Control

Bits	Name	Description
31–16	—	Reserved, should be cleared. This field reserved for future use. It should be cleared.
15–8	μFrame C-mask	Split completion mask. This field (along with the Active and SplitX- state fields in the status byte) is used to determine during which microframes the host controller should execute complete-split transactions. When the criteria for using this field is met, an all-zeros value has undefined behavior. The host controller uses the value of the three low-order bits of the FRINDEX register to index into this bit field. If the FRINDEX register value indexes to a position where the μFrame C-Mask field is a one, this siTD is a candidate for transaction execution. There may be more than one bit in this mask set.
7–0	μFrame S-mask	Split start mask. This field (along with the Active and SplitX-state fields in the Status byte) is used to determine during which microframes the host controller should execute start-split transactions. The host controller uses the value of the three low-order bits of the FRINDEX register to index into this bit field. If the FRINDEX register value indexes to a position where the μFrame S-mask field is a one, then this siTD is a candidate for transaction execution. An all zeros value in this field, in combination with existing periodic frame list has undefined results.

16.5.4.3 siTD Transfer State

DWords 3–6 manage the state of the transfer, as described in [Table 16-48](#).

Table 16-48. siTD Transfer Status and Control

Bits	Name	Description	
31	ioc	Interrupt on complete 0 Do not interrupt when transaction is complete. 1 Do interrupt when transaction is complete. When the host controller determines that the split transaction has completed it will assert a hardware interrupt at the next interrupt threshold.	
30	P	Page select. Indicates which data page pointer should be concatenated with the CurrentOffset field to construct a data buffer pointer 0 Selects Page 0 pointer 1 Selects Page 1 pointer The host controller is not required to write this field back when the siTD is retired (Active bit transitioned from a one to a zero).	
29–26	—	Reserved, should be cleared. This field reserved for future use and should be cleared.	
25–16	Total Bytes to Transfer	This field is initialized by software to the total number of bytes expected in this transfer. Maximum value is 1023 (3FFh)	
15–8	μFrame C-prog-mask	Split complete progress mask. This field is used by the host controller to record which split-completes have been executed.	
7–0	Status	This field records the status of the transaction executed by the host controller for this slot. This field is a bit vector with the following encoding:	
		Status Bits	Definition
		7	Active. Set by software to enable the execution of an isochronous split transaction by the host controller.
		6	ERR. Set by the host controller when an ERR response is received from the companion controller.
		5	Data buffer error. Set by the host controller during status update to indicate that the host controller is unable to keep up with the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (under run). In the case of an under run, the host controller will transmit an incorrect CRC (thus invalidating the data at the endpoint). If an overrun condition occurs, no action is necessary.
		4	Babble detected. Set by the host controller during status update when "babble" is detected during the transaction generated by this descriptor.
		3	Transaction error (XactErr). Set by the host controller during status update in the case where the host did not receive a valid response from the device (Time-out, CRC, Bad PID, etc.). This bit will only be set for IN transactions.
		2	Missed microframe. The host controller detected that a host-induced hold-off caused the host controller to miss a required complete-split transaction.
		1	Split transaction state (SplitXstate). The bit encodings are: 0 Do start split. This value directs the host controller to issue a Start split transaction to the endpoint when a match is encountered in the S-mask. 1 Do complete split. This value directs the host controller to issue a Complete split transaction to the endpoint when a match is encountered in the C-mask.
0	Reserved, should be cleared. Bit reserved for future use and should be cleared.		

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 16-49 describes the siTD buffer pointer page 0.

Table 16-49. siTD Buffer Pointer Page 0 (Plus)

Bits	Name	Description
31–12	Buffer Pointer (Page 0)	Bits 31–12 are 4K page-aligned, physical memory addresses. These bits correspond to physical address bits 31–12 respectively. The field P specifies the current active pointer
11–0	Current Offset	The 12 least-significant bits of the Page 0 pointer is the current byte offset for the current page pointer (as selected with the page indicator bit (P field)). The host controller is not required to write this field back when the siTD is retired (Active bit transitioned from a one to a zero).

Table 16-50 describes the siTD buffer pointer page 1.

Table 16-50. siTD Buffer Pointer Page 1 (Plus)

Bits	Name	Description
31–12	Buffer Pointer (Page 1)	Bits 31–12 are 4K page-aligned, physical memory addresses. These bits correspond to physical address bits 31–12 respectively. The field P specifies the current active pointer
11–5	—	Reserved, should be cleared.
4–3	TP	Transaction position. This field is used with T-count to determine whether to send all, first, middle, or last with each outbound transaction payload. System software must initialize this field with the appropriate starting value. The host controller must correctly manage this state during the lifetime of the transfer. The bit encodings are: 00 All. The entire full-speed transaction data payload is in this transaction (that is, less than or equal to 188 bytes). 01 Begin. This is the first data payload for a full-speed transaction that is greater than 188 bytes. 10 Mid. This is the middle payload for a full-speed OUT transaction that is larger than 188 bytes. 11 End. This is the last payload for a full-speed OUT transaction that was larger than 188 bytes.
2–0	T-Count	Transaction count. Software initializes this field with the number of OUT start-splits this transfer requires. Any value larger than 6 is undefined.

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 16-51 describes the siTD back link pointer.

Table 16-51. siTD Back Link Pointer

Bits	Name	Description
31–5	Back Pointer	A physical memory pointer to an siTD

Table 16-51. siTD Back Link Pointer (continued)

Bits	Name	Description
4–1	—	Reserved, should be cleared. This field is reserved for future use. It should be cleared.
0	T	Terminate 0 siTD Back Pointer field is valid 1 siTD Back Pointer field is not valid

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 20,480 (5×4096) bytes. The structure contains two structure pointers used for queue advancement, a DWord of transfer state, and a five-element array of data buffer pointers. This structure is 32 bytes (or one 32-byte cache line). This data structure must be physically contiguous.

The buffer associated with this transfer must be virtually contiguous. The buffer may start on any byte boundary. A separate buffer pointer list element must be used for each physical page in the buffer, regardless of whether the buffer is physically contiguous.

Host controller updates (host controller writes) to stand-alone qTDs only occur during transfer retirement. References in the following bit field definitions of updates to the qTD are to the qTD portion of a queue head.

Figure 16-40 shows the queue element transfer descriptors.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	offset
Next qTD Pointer																											0000	T	0x00			
Alternate Next qTD Pointer																											0000	T	0x04			
dt 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 16-40. 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 16-52](#) describes the qTD next element transfer pointer.

Table 16-52. qTD Next Element Transfer Pointer (DWord 0)

Bits	Name	Description
31–5	Next qTD Pointer	This field contains the physical memory address of the next qTD to be processed and corresponds to memory address signals [31:5], respectively.
4–1	—	Reserved, should be cleared. These bits are reserved and their value has no effect on operation.
0	T	Terminate. Indicates to the host controller that there are no more valid entries in the queue. 0 Pointer is valid (points to a valid transfer element descriptor) 1 Pointer is invalid

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 16-53](#) describes the alternate qTD next element transfer pointer.

Table 16-53. qTD Alternate Next Element Transfer Pointer (DWord 1)

Bits	Name	Description
31–5	Alternate Next qTD Pointer	This field contains the physical memory address of the next qTD to be processed in the event that the current qTD execution encounters a short packet (for an IN transaction). The field corresponds to memory address signals [31:5], respectively.
4–1	—	Reserved, should be cleared. These bits are reserved and their value has no effect on operation.
0	T	Terminate. Indicates to the host controller that there are no more valid entries in the queue. 0 Pointer is valid (points to a valid transfer element descriptor) 1 Pointer is invalid

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 16-54](#) reference fields are defined in the queue head. See [Section 16.5.6, “Queue Head,”](#) for more information on these fields.

Table 16-54. qTD Token (DWord 2)

Bits	Name	Description
31	dt	Data toggle. This is the data toggle sequence bit. The use of this bit depends on the setting of the Data Toggle Control bit in the queue head.
30–16	Total Bytes to Transfer	Total bytes to transfer. This field specifies the total number of bytes to be moved with this transfer descriptor. This field is decremented by the number of bytes actually moved during the transaction, only on the successful completion of the transaction. The maximum value software may store in this field is $5 \times 4\text{K}$ (0x5000). This is the maximum number of bytes 5 page pointers can access. If the value of this field is zero when the host controller fetches this transfer descriptor (and the active bit is set), the host controller executes a zero-length transaction and retires the transfer descriptor. It is not a requirement for OUT transfers that total bytes to transfer be an even multiple of QH[Maximum Packet Length]. If software builds such a transfer descriptor for an OUT transfer, the last transaction will always be less than QH[Maximum Packet Length]. Although it is possible to create a transfer up to 20K this assumes the page is 0. When the offset cannot be predetermined, crossing past the 5th page can be guaranteed by limiting the total bytes to 16K. Therefore, the maximum recommended transfer is 16K (0x4000).
15	ioc	Interrupt on complete. If this bit is set, the host controller should issue an interrupt at the next interrupt threshold when this qTD is completed.
14–12	C_Page	Current rage. This field is used as an index into the qTD buffer pointer list. Valid values are in the range 0x0 to 0x4. The host controller is not required to write this field back when the qTD is retired.

Table 16-54. qTD Token (DWord 2) (continued)

Bits	Name	Description	
11–10	Cerr	Error counter. 2-bit down counter that keeps track of the number of consecutive errors detected while executing this qTD. If this field is programmed with a non-zero value during setup, the host controller decrements the count and writes it back to the qTD if the transaction fails. If the counter counts from one to zero, the host controller marks the qTD inactive, sets the Halted bit to a one, and error status bit for the error that caused Cerr to decrement to zero. An interrupt is generated if USBINTR[UEE] is set. If the host controller driver (HCD) software programs this field to zero during setup, the host controller will not count errors for this qTD and there is no limit on the retries of this qTD. Note that write-backs of intermediate execution state are to the queue head overlay area, not the qTD.	
		Error	Decrement Counter
		Transaction Error	Yes
		Data Buffer Error	No. Data buffer errors are host problems. They don't count against the device's retries. Note that software must not program Cerr to a value of zero when the EPS field is programmed with a value indicating a full- or low-speed device. This combination could result in undefined behavior.
		Stalled	No. Detection of babble or stall automatically halts the queue head. Thus, count is not decremented
		Babble Detected	No. Detection of babble or stall automatically halts the queue head. Thus, count is not decremented
		No Error	No. If the EPS field indicates a HS device or the queue head is in the asynchronous schedule (and PIDCode indicates an IN or OUT) and a bus transaction completes and the host controller does not detect a transaction error, then the host controller should reset Cerr to extend the total number of errors for this transaction. For example, Cerr should be reset with maximum value (0b11) on each successful completion of a transaction. The host controller must never reset this field if the value at the start of the transaction is 0b00.
9–8	PID Code	This field is an encoding of the token, which should be used for transactions associated with this transfer descriptor. Encodings are: 00 OUT Token generates token (E1H) 01 IN Token generates token (69H) 10 SETUP Token generates token (2DH) (undefined if endpoint is an Interrupt transfer type, for example. μ Frame S-mask field in the queue head is non-zero.) 11 Reserved, should be cleared	

Table 16-54. qTD Token (DWord 2) (continued)

Bits	Name	Description														
7-0	Status	This field is used by the host controller to communicate individual command execution states back to the host controller driver (HCD) software. This field contains the status of the last transaction performed on this qTD. The bit encodings are:														
		<table border="1"> <thead> <tr> <th data-bbox="428 380 659 443">Bits</th> <th data-bbox="659 380 1479 443">Status Field Description</th> </tr> </thead> <tbody> <tr> <td data-bbox="428 443 659 520">7</td> <td data-bbox="659 443 1479 520">Active. Set by software to enable the execution of transactions by the host controller.</td> </tr> <tr> <td data-bbox="428 520 659 716">6</td> <td data-bbox="659 520 1479 716">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 data-bbox="428 716 659 936">5</td> <td data-bbox="659 716 1479 936">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 data-bbox="428 936 659 1100">4</td> <td data-bbox="659 936 1479 1100">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 data-bbox="428 1100 659 1236">3</td> <td data-bbox="659 1100 1479 1236">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 data-bbox="428 1236 659 1425">2</td> <td data-bbox="659 1236 1479 1425">Missed microframe. This bit is ignored unless the QH[EPS] field indicates a full- or low-speed endpoint and the queue head is in the periodic list. This bit is set when the host controller detected that a host-induced hold-off caused the host controller to miss a required complete-split transaction. If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.</td> </tr> </tbody> </table>	Bits	Status Field Description	7	Active. Set by software to enable the execution of transactions by the host controller.	6	Halted. Set by the host controller during status updates to indicate that a serious error has occurred at the device/endpoint addressed by this qTD. This can be caused by babble, the error counter counting down to zero, or reception of the STALL handshake from the device during a transaction. Any time that a transaction results in the Halted bit being set, the Active bit is also cleared.	5	Data buffer error. Set by the host controller during status update to indicate that the host controller is unable to keep up with the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (under run). If an overrun condition occurs, the host controller will force a time-out condition on the USB, invalidating the transaction at the source. If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.	4	Babble detected. Set by the host controller during status update when babble is detected during the transaction. In addition to setting this bit, the host controller also sets the Halted bit to a one. Since babble is considered a fatal error for the transfer, setting the Halted bit to a one insures that no more transactions occur because of this descriptor.	3	Transaction error (XactErr). Set by the host controller during status update in the case where the host did not receive a valid response from the device (time-out, CRC, bad PID). If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.	2	Missed microframe. This bit is ignored unless the QH[EPS] field indicates a full- or low-speed endpoint and the queue head is in the periodic list. This bit is set when the host controller detected that a host-induced hold-off caused the host controller to miss a required complete-split transaction. If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.
	Bits	Status Field Description														
	7	Active. Set by software to enable the execution of transactions by the host controller.														
	6	Halted. Set by the host controller during status updates to indicate that a serious error has occurred at the device/endpoint addressed by this qTD. This can be caused by babble, the error counter counting down to zero, or reception of the STALL handshake from the device during a transaction. Any time that a transaction results in the Halted bit being set, the Active bit is also cleared.														
	5	Data buffer error. Set by the host controller during status update to indicate that the host controller is unable to keep up with the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (under run). If an overrun condition occurs, the host controller will force a time-out condition on the USB, invalidating the transaction at the source. If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.														
	4	Babble detected. Set by the host controller during status update when babble is detected during the transaction. In addition to setting this bit, the host controller also sets the Halted bit to a one. Since babble is considered a fatal error for the transfer, setting the Halted bit to a one insures that no more transactions occur because of this descriptor.														
	3	Transaction error (XactErr). Set by the host controller during status update in the case where the host did not receive a valid response from the device (time-out, CRC, bad PID). If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.														
2	Missed microframe. This bit is ignored unless the QH[EPS] field indicates a full- or low-speed endpoint and the queue head is in the periodic list. This bit is set when the host controller detected that a host-induced hold-off caused the host controller to miss a required complete-split transaction. If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.															

Table 16-54. qTD Token (DWord 2) (continued)

Bits	Name	Description
1		<p>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:</p> <p>0 Do start split. This value directs the host controller to issue a start split transaction to the endpoint.</p> <p>1 Do complete split. This value directs the host controller to issue a Complete split transaction to the endpoint.</p>
0		<p>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:</p> <p>0 Do OUT. This value directs the host controller to issue an OUT PID to the endpoint.</p> <p>1 Do Ping. This value directs the host controller to issue a PING PID to the endpoint.</p> <p>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.</p>

16.5.5.4 qTD Buffer Page Pointer List

The last five DWords of a queue element transfer descriptor make up an array of physical memory address pointers. These pointers reference the individual pages of a data buffer.

System software initializes the Current Offset field to the starting offset into the current page, where current page is selected with the value in the C_Page field.

Table 16-55 describes the qTD buffer pointer.

Table 16-55. qTD Buffer Pointer

Bits	Name	Description
31–12	Buffer Pointer (page <i>n</i>)	Each element in the list is a 4K page-aligned physical memory address. The lower 12 bits in each pointer are reserved (except for the first one), as each memory pointer must reference the start of a 4K page. The field C_Page specifies the current active pointer. When the transfer element descriptor is fetched, the starting buffer address is selected using C_Page (similar to an array index to select an array element). If a transaction spans a 4K buffer boundary, the host controller must detect the page-span boundary in the data stream, increment C_Page and advance to the next buffer pointer in the list, and conclude the transaction via the new buffer pointer.
11–0	Current Offset (Page 0)/ — (Pages 1–4)	Reserved in all pointers except the first one (that is, Page 0). The host controller should ignore all reserved bits. For the page 0 current offset interpretation, this field is the byte offset into the current page (as selected by C_Page). The host controller is not required to write this field back when the qTD is retired. Software should ensure the reserved fields are initialized to zeros.

16.5.6 Queue Head

Figure 16-41 shows the queue head structure.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	offset
Queue Head Horizontal Link Pointer																											00	Typ	T	0x00		
RL		C	Maximum Packet Length				H	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 16-41. 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 16-56 describes the queue head.

Table 16-56. Queue Head DWord 0

Bits	Name	Description
31–5	QHLP	Queue head horizontal link pointer. This field contains the address of the next data object to be processed in the horizontal list and corresponds to memory address signals [31:5], respectively.
4–3	—	Reserved, should be cleared. These bits must be written as zeros.

Table 16-56. Queue Head DWord 0 (continued)

Bits	Name	Description
2–1	Typ	Indicates to the hardware whether the item referenced by the link pointer is an iTD, siTD or a QH. This allows the host controller to perform the proper type of processing on the item after it is fetched. 00 iTD (isochronous transfer descriptor) 01 QH (queue head) 10 siTD (split transaction isochronous transfer descriptor) 11 FSTN (frame span traversal node)
0	T	Terminate. 1 Last QH (pointer is invalid). 0 Pointer is valid. If the queue head is in the context of the periodic list, a one bit in this field indicates to the host controller that this is the end of the periodic list. This bit is ignored by the host controller when the queue head is in the asynchronous schedule. Software must ensure that queue heads reachable by the host controller always have valid horizontal link pointers.

16.5.6.2 Endpoint Capabilities/Characteristics

The second and third DWords of a queue head specify static information about the endpoint. This information does not change over the lifetime of the endpoint. There are three types of information in this region:

- Endpoint characteristics. These are the USB endpoint characteristics, which include addressing, maximum packet size, and endpoint speed.
- Endpoint capabilities. These are adjustable parameters of the endpoint. They affect how the endpoint data stream is managed by the host controller.
- Split transaction characteristics. This data structure manages full- and low-speed data streams for bulk, control, and interrupt with split transactions to USB 2.0 Hub transaction translator. Additional fields exist for addressing the hub and scheduling the protocol transactions (for periodic).

The host controller must not modify the bits in this region.

[Table 16-57](#) and [Table 16-58](#) describe the endpoint characteristics.

Table 16-57. Endpoint Characteristics: Queue Head DWord 1

Bits	Name	Description
31–28	RL	Nak count reload. This field contains a value, which is used by the host controller to reload the Nak Counter field.
27	C	Control endpoint flag. If the QH[EPS] field indicates the endpoint is not a high-speed device, and the endpoint is a control endpoint, then software must set this bit to a one. Otherwise, it should always set this bit to a zero.
26–16	Maximum Packet Length	This directly corresponds to the maximum packet size of the associated endpoint (wMaxPacketSize). The maximum value this field may contain is 0x400 (1024).
15	H	Head of reclamation list flag. This bit is set by system software to mark a queue head as being the head of the reclamation list.

Table 16-57. Endpoint Characteristics: Queue Head DWord 1 (continued)

Bits	Name	Description
14	dtc	Data toggle control (DTC). Specifies where the host controller should get the initial data toggle on an overlay transition. 0 Ignore DT bit from incoming qTD. Host controller preserves DT bit in the queue head. 1 Initial data toggle comes from incoming qTD DT bit. Host controller replaces DT bit in the queue head from the DT bit in the qTD.
13–12	EPS	Endpoint speed. This is the speed of the associated endpoint. 00 Full-speed (12 Mbps) 01 Low-speed (1.5 Mbps) 10 High-speed (480 Mbps) 11 Reserved, should be cleared This field must not be modified by the host controller.
11–8	EndPt	Endpoint number. Selects the particular endpoint number on the device serving as the data source or sink.
7	I	Inactivate on next transaction. This bit is used by system software to request that the host controller set the Active bit to zero. This field is only valid when the queue head is in the periodic schedule and the EPS field indicates a full- or low-speed endpoint. Setting this bit when the queue head is in the asynchronous schedule or the EPS field indicates a high-speed device yields undefined results.
6–0	Device Address	Selects the specific device serving as the data source or sink.

Table 16-58. Endpoint Capabilities: Queue Head DWord 2

Bits	Name	Description
31–30	Mult	High-bandwidth pipe multiplier. This field is a multiplier used to key the host controller as the number of successive packets the host controller may submit to the endpoint in the current execution. The host controller makes the simplifying assumption that software properly initializes this field (regardless of location of queue head in the schedules or other run time parameters). 00 Reserved, should be cleared. A zero in this field yields undefined results. 01 One transaction to be issued for this endpoint per microframe 10 Two transactions to be issued for this endpoint per microframe 11 Three transactions to be issued for this endpoint per microframe
29–23	Port Number	This field is ignored by the host controller unless the EPS field indicates a full- or low-speed device. The value is the port number identifier on the USB 2.0 hub (for hub at device address Hub Addr below), below which the full- or low-speed device associated with this endpoint is attached. This information is used in the split-transaction protocol.
22–16	Hub Addr	This field is ignored by the host controller unless the EPS field indicates a full- or low-speed device. The value is the USB device address of the USB 2.0 hub below which the full- or low-speed device associated with this endpoint is attached. This field is used in the split-transaction protocol.

Table 16-58. Endpoint Capabilities: Queue Head DWord 2 (continued)

Bits	Name	Description
15–8	μFrame C-mask	This field is ignored by the host controller unless the EPS field indicates this device is a low- or full-speed device and this queue head is in the periodic list. This field (along with the Active and SplitX-state fields) is used to determine during which microframes the host controller should execute a complete-split transaction. When the criteria for using this field are met, a zero value in this field has undefined behavior. This field is used by the host controller to match against the three low-order bits of the FRINDEX register. If the FRINDEX register bits decode to a position where the μFrame C- mask field is a one, then this queue head is a candidate for transaction execution. There may be more than one bit in this mask set.
7–0	μFrame S-mask	Interrupt schedule mask. This field is used for all endpoint speeds. Software should set this field to a zero when the queue head is on the asynchronous schedule. A non-zero value in this field indicates an interrupt endpoint. The host controller uses the value of the three low-order bits of the FRINDEX register as an index into a bit position in this bit vector. If the μFrame S-mask field has a one at the indexed bit position then this queue head is a candidate for transaction execution. If the EPS field indicates the endpoint is a high-speed endpoint, then the transaction executed is determined by the PID_Code field contained in the execution area. This field is also used to support split transaction types: Interrupt (IN/OUT). This condition is true when this field is non-zero and the EPS field indicates this is either a full- or low-speed device. A zero value in this field, in combination with existing in the periodic frame list has undefined results.

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 16-58 describes the current qTD link pointer.

Table 16-59. Current qTD Link Pointer

Bits	Name	Description
31–5	Current qTD Pointer	Current element transaction descriptor link pointer. This field contains the address Of the current transaction being processed in this queue and corresponds to memory address signals [31:5], respectively.
4–0	—	Reserved, should be cleared. These bits are ignored by the host controller when using the value as an address to write data. The actual value may vary depending on the usage.

The DWords 4–11 of a queue head are the transaction overlay area. This area has the same base structure as a queue element transfer descriptor. The queue head utilizes the reserved fields of the page pointers to implement tracking the state of split transactions.

This area is characterized as an overlay because when the queue is advanced to the next queue element, the source queue element is merged onto this area. This area serves an execution cache for the transfer.

Table 16-60 describes the host-controller rules for bits in overlay.

Table 16-60. Host-Controller Rules for Bits in Overlay (DWords 5, 6, 8, and 9)

DWord	QH Offset	Bits	Name	Description
5	0x14	4–1	NakCnt	Nak counter—RW. This field is a counter the host controller decrements whenever a transaction for the endpoint associated with this queue head results in a Nak or Nyet response. This counter is reloaded from RL before a transaction is executed during the first pass of the reclamation list (relative to an Asynchronous List Restart condition). It is also loaded from RL during an overlay.
6	0x18	31	dt	Data toggle. The Data toggle control controls whether the host controller preserves this bit when an overlay operation is performed.
6	0x18	15	ioc	Interrupt on complete. The ioc control bit is always inherited from the source qTD when the overlay operation is performed.
6	0x18	11–10	Cerr	Error counter. Copied from the qTD during the overlay and written back during queue advancement.
6	0x18	0	Status[0]	Ping state (P)/ERR. If the EPS field indicates a high-speed endpoint, then this field should be preserved during the overlay operation.
8	0x20	7–0	C-prog-mask	Split-transaction complete-split progress. Initialized to zero during any overlay. This field is used to track the progress of an interrupt split-transaction.
9	0x24	11–5	S-bytes	Software must ensure that the S-bytes field in a qTD is zero before activating the qTD. Keeps track of the number of bytes sent or received during an IN or OUT split transaction.
9	0x24	4–0	FrameTag	Split-transaction frame tag. Initialized to zero during any overlay. This field is used to track the progress of an interrupt split-transaction.

16.5.7 Periodic Frame Span Traversal Node (FSTN)

The periodic frame span traversal node (FSTN) data structure, shown in Figure 16-42, is to be used only for managing full- and low-speed transactions that span a host-frame boundary. Software must not use an FSTN in the asynchronous schedule. An FSTN in the asynchronous schedule results in undefined behavior.

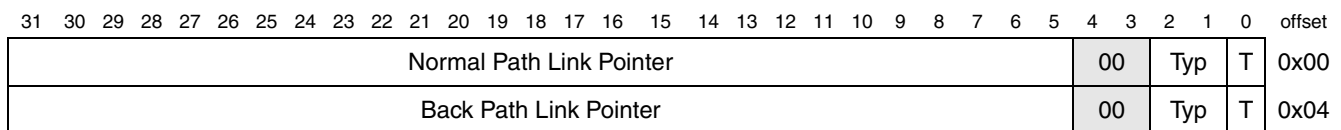


Figure 16-42. Frame Span Traversal Node Structure

NOTE

The host controller performs only read operations to the FSTN data structure.

16.5.7.1 FSTN 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 16-61](#) describes the FSTN normal path pointer.

Table 16-61. FSTN Normal Path Pointer

Bits	Name	Description
31–5	NPLP	Normal path link pointer. Contains the address of the next data object to be processed in the periodic list and corresponds to memory address signals [31:5], respectively.
4–3	—	Reserved, should be cleared. These bits must be written as 0s.
2–1	Typ	Indicates to the host controller whether the item referenced is a iTD/siTD, QH, or FSTN. This allows the host controller to perform the proper type of processing on the item after it is fetched. 00 iTD (isochronous transfer descriptor) 01 QH (queue head) 10 siTD (split transaction isochronous transfer descriptor) 11 FSTN (frame span traversal node)
0	T	Terminate. 0 Link pointer is valid. 1 Link pointer field is not valid.

16.5.7.2 FSTN Back Path Link Pointer

The second DWord of an FSTN 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 16-62](#) describes the FSTN back path link pointer.

Table 16-62. FSTN Back Path Link Pointer

Bits	Name	Description
31–5	BPLP	Back path link pointer. Contains the address of a queue head. This field corresponds to memory address signals [31:5], respectively.
4–3	—	Reserved, should be cleared. These bits must be written as 0s.
2–1	Typ	Software must ensure this field is set to indicate the target data structure is a Queue Head (01). Any other value in this field yields undefined results.
0	T	Terminate. 0 Link pointer is valid (that is, the host controller may use bits 31–5 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 as a valid memory address). This value also indicates that this FSTN is a Restore indicator.

16.6 Host Operations

The general operational model for the USB DR module in host mode is defined by the EHCI specification. The EHCI specification describes the register-level interface for a host controller for the USB Revision 2.0. It includes a description of the hardware/software interface between system software and host

controller hardware. Information concerning the initialization of the USB module is included in the following section; however, the full details of the EHCI specification are beyond the scope of this document.

16.6.1 Host Controller Initialization

After initial power-on or host controller reset (hardware or through USBCMD[RST]), all of the operational registers are at their default values.

To configure the external ULPI PHY the following initialization sequence is required:

1. The UTMI PHY should remain disabled if the ULPI is being used.
2. Set the CONTROL[PHY_CLK_SEL] bits to select the ULPI PHY as the source of USB controller PHY clock.
3. Wait for PHY clock to become valid. This can be determined by polling the CONTROL[PHY_CLK_VALID] status bit. Note that this bit is not valid once the CONTROL[USB_EN] bit is set

Once the PHY clock is valid the user can proceed to the host controller initialization phase.

In order to initialize the USB DR module, software should perform the following steps

1. Set the controller mode to host mode. Optionally set USBMODE[SDIS] (streaming disable)

NOTE

Transitioning from device mode to host mode requires a host controller reset before modifying USBMODE.

2. Optionally modify the BURSTSIZE register.
3. Program the PTS field of the PORTSC register to 2'b10 to indicate the use of ULPI PHY.
4. Set CONTROL[USB_EN].
5. Write the appropriate value to the USBINTR register to enable the appropriate interrupts.
6. Write the base address of the periodic frame list to the PERIODICLIST BASE register. If there are no work items in the periodic schedule, all elements of the periodic frame list should have their T-Bits set.
7. Write the USBCMD register to set the desired interrupt threshold, frame list size (if applicable) and turn on the controller by setting the RS bit.

At this point, the USB DR module is up and running and the port registers begin reporting device connects. System software can enumerate a port through the reset process (where the port is in the enabled state). At this point, the port is active with SOFs occurring down the enabled high-speed ports, but the schedules have not yet been enabled. The EHCI host controller will not transmit SOFs to enabled Full- or Low-speed ports.

In order to communicate with devices via the asynchronous schedule, system software must write the ASYNDLISTADDR register with the address of a control or bulk queue head. Software must then enable the asynchronous schedule by writing a one to USBCMD[ASE]. In order to communicate with devices via

the periodic schedule, system software must enable the periodic schedule by writing a one to USBCMD[PSE]. Note that the schedules can be turned on before the first port is reset (and enabled).

Any time the USBCMD register is written, system software must ensure the appropriate bits are preserved, depending on the intended operation.

16.6.2 Power Port

The HCSPARAMS[PPC] bit indicates whether the USB 2.0 host controller has port power control. When the PPC bit is set, the host controller supports port power switches. Each available switch has an output enable. PPE is controlled based on the state of the combination bits —PPC bit, EHCI Configured (CF)-bit and individual Port Power (PP) bits. The Configured Flag and Port Power Control bits are always 1 in Host Mode. The PPE always follows the state of Port Power (PP) bit that is, if PP is 0, PPE will be 0 and if PP is 1, PPE will be 1.

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. The EHCI PORTSC register has an over-current status and over-current change bit. The functionality of these bits is specified in the USB Specification Revision 2.0.

The over current detection and limiting logic resides outside the DR logic. The over-current condition effects the following bits in the PORTSC register on the EHCI port:

- Over-current active bit (OCA) is set. When the over-current condition goes away, the OCA will transition from a one to a zero.
- Over-current change bit (OCC) is set. On every transition of OCA, the controller will set OCC to a one. Software sets OCC to a zero by writing a one to this bit.
- Port enabled/disabled bit (PE) is cleared.
- 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. In addition, if the Port Change Interrupt Enable bit, USBINTR[PCE], is a one, the controller issues an interrupt to the system. Refer to [Table 16-63](#) 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 through software initiation. Other control mechanisms are provided to parameterize the host controller's response (or sensitivity) to external resume events. In this discussion, host-initiated, or software-initiated resumes are called Resume Events/Actions; bus-initiated resume events are called wake-up events. The classes of wakeup events are:

- Remote-wakeup enabled device asserts resume signaling. In similar kind to USB 2.0 hubs, when in host mode the host controller responds to explicit device resume signaling and wake up the system (if necessary).
- Port connect and disconnect and over-current events. Sensitivity to these events can be turned on or off using the port control bits in the PORTSC register.

Selective suspend is a feature supported by the PORTSC register. It is used to place specific ports into a suspend mode. This feature is used as a functional component for implementing the appropriate power management policy implemented in a particular operating system. When system software intends to suspend the bus, it should suspend the enabled port, then shut off the controller by setting the USBCMD[RS] to a zero.

When a wake event occurs the system will resume operation and system software must set the RS bit to a one and resume the suspended port.

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 microframe or frame boundary occurs. If evaluated immediately, the port is not suspended until the current transaction (if one is executing) completes. Therefore, there may be several microframes of activity on the port until the host controller evaluates the Suspend bit. The host controller must evaluate the Suspend bit at least every frame boundary.

System software can initiate a resume on the suspended port by writing a one to PORTSC[FPR]. Software should not attempt to resume a port unless the port reports that it is in the suspended state. If system software sets PORTSC[FPR] when the port is not in the suspended state, the resulting behavior is undefined. In order to assure proper USB device operation, software must wait for at least 10 milliseconds after a port indicates that it is suspended (Suspend bit is a one) before initiating a port resume through PORTSC[FPR]. When PORTSC[FPR] is set, the host controller sends resume signaling down the port. System software times the duration of the resume (nominally 20 milliseconds) then clears PORTSC[FPR]. When the host controller receives the write to transition PORTSC[FPR] to zero, it completes the resume sequence as defined in the USB specification, and clears both PORTSC[FPR] and PORTSC[SUSP]. Software-initiated port resumes do not cause an interrupt if USBINTR[PCE] (port change interrupt enable) is a one. When a wake event occurs on a suspended port, the resume signaling is detected by the port and the resume is reflected downstream within 100 μ sec. The port's PORTSC[FPR] bit is set. If USBINTR[PCE] is a one, the host controller issues a hardware interrupt.

System software observes the resume event on the port, delays a port resume time (nominally 20 milliseconds), then terminates the resume sequence by clearing PORTSC[FPR] in the port. The host controller receives the write of zero to PORTSC[FPR], terminates the resume sequence and clears PORTSC[FPR] and PORTSC[SUSP]. Software can determine that the port is enabled (not suspended) by sampling the PORTSC register and observing that the SUSP and FPR bits are zero. Software must ensure that the host controller is running (that is, USBSTS[HCH] is a zero), before terminating a resume by

clearing the port's PORTSC[FPR] bit. If HCH is a one when PORTSC[FPR] is cleared, then SOFs will not occur down the enabled port and the device will return to suspend mode in a maximum of 10 milliseconds.

Table 16-63 summarizes the wake-up events. If USBINTR[PCE] (port change interrupt enable) is a one, the host controller also generates an interrupt on the resume event.

Table 16-63. Behavior During Wake-Up Events

Port Status and Signaling Type	Signaled Port Response	Device State	
		D0	not D0
Port disabled, resume K-State received	No effect	N/A	N/A
Port suspended, resume K-State received	Resume reflected downstream on signaled port. PORTSC[FPR] is set.	[1], [2]	[2]
Port is enabled, disabled or suspended, and the port's bit, PORTSC[WKDS], is set. A disconnect is detected.	Depending on the initial port state, the PORTSC Connect (CCS) and Enable (PE) status bits are cleared, and the Connect Change status bit (CSC) is set.	[1], [2]	[2]
Port is enabled, disabled or suspended, and the port's bit, PORTSC[WKDS], is cleared. A disconnect is detected.	Depending on the initial port state, the PORTSC Connect (CCS) and Enable (PE) status bits are cleared, and the Connect Change status bit (CSC) is set.	[1], [3]	[3]
Port is not connected and the port's PORTSC[WKCN] bit is a one. A connect is detected.	PORTSC Connect Status (CCS) and Connect Status Change (CSC) bits are set.	[1], [2]	[2]
Port is not connected and the port's PORTSC[WKCN] bit is a zero. A connect is detected.	PORTSC Connect Status (CCS) and Connect Status Change (CSC) bits are set.	[1], [3]	[3]
Port is connected and the port's PORTSC[WKOC] bit is a one. An over-current condition occurs.	PORTSC Over-current Active (OCA), Over-current Change (OCC) bits are set. If Port Enable/Disable bit (PE) is a one, it is cleared.	[1], [2]	[2]
Port is connected and the port's PORTSC[WKOC] bit is a zero. An over-current condition occurs.	PORTSC Over-current Active (OCA), Over-current Change (OCC) bits are set. If Port Enable/Disable bit (PE) is a one, it is cleared.	[1], [3]	[3]

¹ Hardware interrupt issued if USBINTR[PCE] (port change interrupt enable) is set.

² PME# asserted if enabled (Note: PME Status must always be set).

³ PME# not asserted.

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 microframe, if the periodic schedule is enabled (see) then the host controller must execute from the periodic schedule before executing from the asynchronous schedule. It will only execute from the asynchronous schedule after it encounters the end of the periodic schedule. The host controller traverses the periodic schedule by constructing an array offset reference from the PERIODICLISTBASE and the FRINDEX registers (see Figure 16-43). It fetches the element and begins traversing the graph of linked schedule data structures.

The end of the periodic schedule is identified by a next link pointer of a schedule data structure having its T-bit set. When the host controller encounters a T-Bit set during a horizontal traversal of the periodic list, it interprets this as an End-Of-Periodic-List mark. This causes the host controller to cease working on the periodic schedule and transitions immediately to traversing the asynchronous schedule. Once this transition is made, the host controller executes from the asynchronous schedule until the end of the microframe.

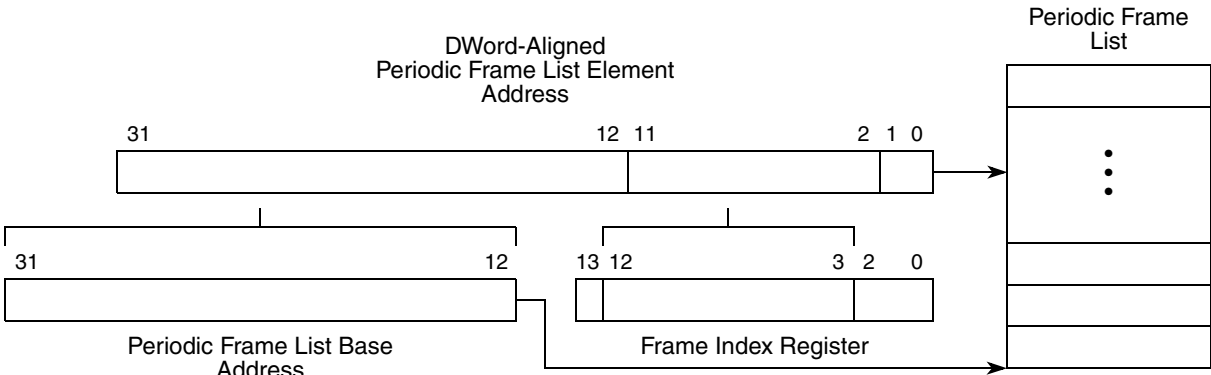


Figure 16-43. Derivation of Pointer into Frame List Array

When the host controller determines that it is time to execute from the asynchronous list, it uses the operational register ASYNCLISTADDR to access the asynchronous schedule, as shown in Figure 16-44.

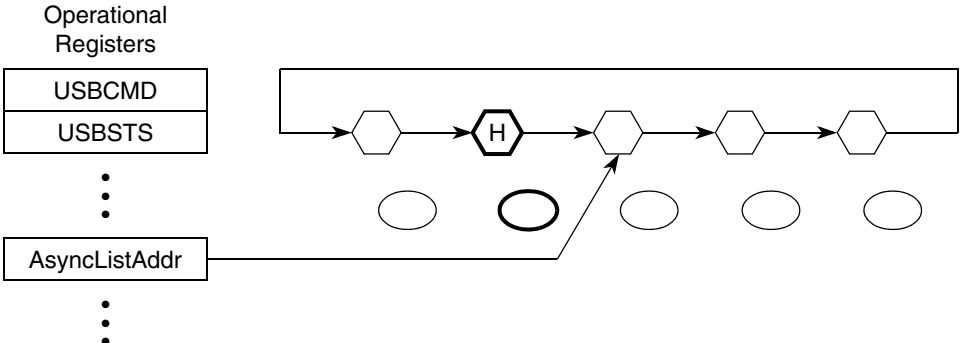


Figure 16-44. General Format of Asynchronous Schedule List

The ASYNCLISTADDR register contains a physical memory pointer to the next queue head. When the host controller makes a transition to executing the asynchronous schedule, it begins by reading the queue head referenced by the ASYNCLISTADDR register. Software must set queue head horizontal pointer T-bits to a zero for queue heads in the asynchronous schedule.

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 microframe pipeline (see start- (SS) and complete- (CS) splits illustrated in [Figure 16-45](#)). A simple, direct projection of the frame boundary model into the host controller interface schedule architecture creates tension (complexity for both hardware and software) between the frame boundaries and the scheduling mechanisms required to service the full- and low-speed transaction translator periodic pipelines.

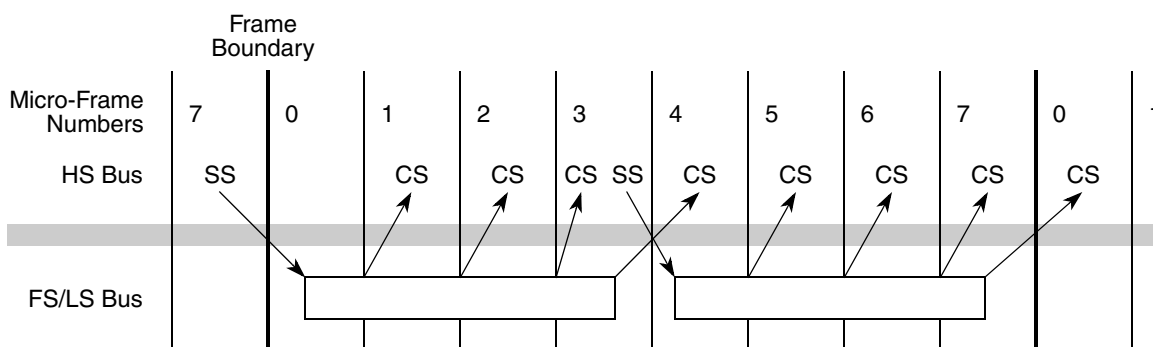


Figure 16-45. Frame Boundary Relationship Between HS Bus and FS/LS Bus

The simple projection, as [Figure 16-45](#) illustrates, introduces frame-boundary wrap conditions for scheduling on both the beginning and end of a frame. In order to reduce the complexity for hardware and software, the host controller is required to implement a one microframe phase shift for its view of frame boundaries. The phase shift eliminates the beginning of frame and frame-wrap scheduling boundary conditions.

The implementation of this phase shift requires that the host controller use one register value for accessing the periodic frame list and another value for the frame number value included in the SOF token. These two values are separate, but tightly coupled. The periodic frame list is accessed via the Frame List Index Register (FRINDEX). Bits FRINDEX[2–0], represent the microframe number. The SOF value is coupled to the value of FRINDEX[13–3]. Both FRINDEX[13–3] and the SOF value are incremented based on FRINDEX[2–0]. It is required that the SOF value be delayed from the FRINDEX value by one microframe. The one microframe delay yields a host controller periodic schedule and bus frame boundary relationship as illustrated in [Figure 16-46](#). This adjustment allows software to trivially schedule the periodic start and complete-split transactions for full-and low-speed periodic endpoints, using the natural alignment of the periodic schedule interface.

[Figure 16-46](#) illustrates how periodic schedule data structures relate to schedule frame boundaries and bus frame boundaries. To aid the presentation, two terms are defined. The host controller's view of the

1-millisecond boundaries is called H-Frames. The high-speed bus's view of the 1-millisecond boundaries is called B-Frames.

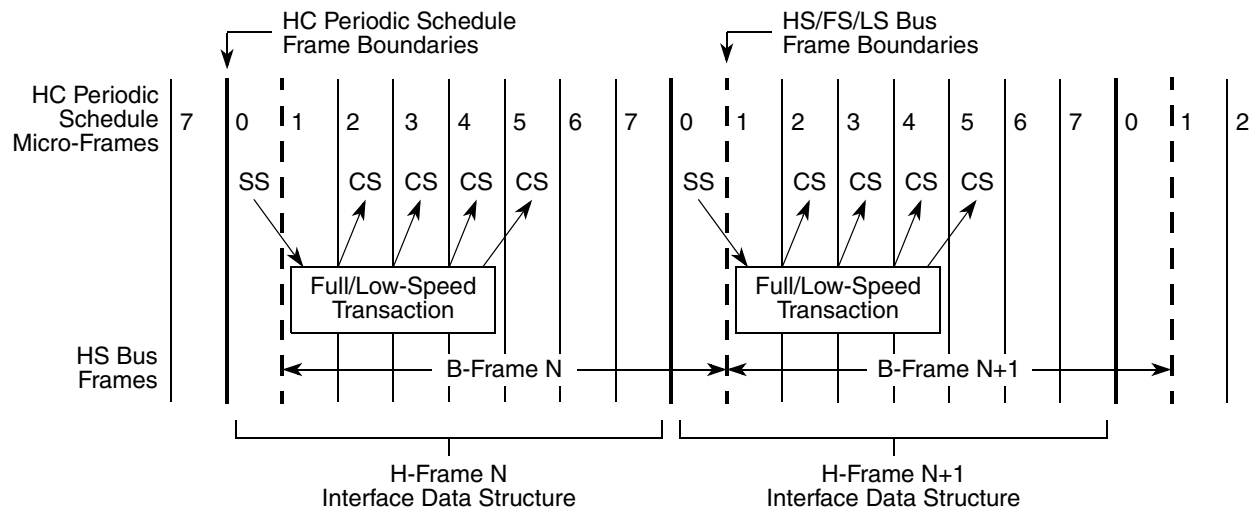


Figure 16-46. Relationship of Periodic Schedule Frame Boundaries to Bus Frame Boundaries

H-Frame boundaries for the host controller correspond to increments of $FRINDEX[13-3]$. Microframe numbers for the H-Frame are tracked by $FRINDEX[2-0]$. B-Frame boundaries are visible on the high-speed bus via changes in the SOF token's frame number. Microframe numbers on the high-speed bus are only derived from the SOF token's frame number (that is, the high-speed bus will see eight SOFs with the same frame number value). H-Frames and B-Frames have the fixed relationship (that is, B-Frames lag H-Frames by one microframe time) illustrated in [Figure 16-46](#). The host controller's periodic schedule is naturally aligned to H-Frames. Software schedules transactions for full- and low-speed periodic endpoints relative the H-Frames. The result is these transactions execute on the high-speed bus at exactly the right time for the USB 2.0 hub periodic pipeline. As described in [Section 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 microframe count. [Table 16-64](#) illustrates the required relationship between the value of $FRINDEX$ and the value of SOFV. This lag behavior can be accomplished by incrementing $FRINDEX[13-3]$ based on carry-out on the 7 to 0 increment of $FRINDEX[2-0]$ and incrementing SOFV based on the transition of 0 to 1 of $FRINDEX[2-0]$.

Software is allowed to write to FRINDEX. Section 16.3.2.4, “Frame Index Register (FRINDEX),” provides the requirements that software should adhere when writing a new value in FRINDEX.

Table 16-64. Operation of FRINDEX and SOFV (SOF Value Register)

Current			Next		
FRINDEX[13–3]	SOFV	FRINDEX[2–0]	FRINDEX[13–3]	SOFV	FRINDEX[2–0]
N	N	111	N+1	N	000
N+1	N	000	N+1	N+1	001
N+1	N+1	001	N+1	N+1	010
N+1	N+1	010	N+1	N+1	011
N+1	N+1	011	N+1	N+1	100
N+1	N+1	100	N+1	N+1	101
N+1	N+1	101	N+1	N+1	110
N+1	N+1	110	N+1	N+1	111

16.6.7 Periodic Schedule

The periodic schedule traversal is enabled or disabled through USBCMD[PSE] (periodic schedule enable). If USBCMD[PSE] is cleared, then the host controller simply does not try to access the periodic frame list via the PERIODICLISTBASE register. Likewise, when USBCMD[PSE] is a one, then the host controller does use the PERIODICLISTBASE register to traverse the periodic schedule. The host controller will not react to modifications to USBCMD[PSE] immediately. In order to eliminate conflicts with split transactions, the host controller evaluates USBCMD[PSE] only when FRINDEX[2–0] is zero. System software must not disable the periodic schedule if the schedule contains an active split transaction work item that spans the 0b000 microframe. These work items must be removed from the schedule before USBCMD[PSE] is cleared. USBSTS[PS] (periodic schedule status) indicates status of the periodic schedule. System software enables (or disables) the periodic schedule by setting (or clearing) USBCMD[PSE]. Software then can poll USBSTS[PS] to determine when the periodic schedule has made the desired transition. Software must not modify USBCMD[PSE] unless the value of USBCMD[PSE] equals that of USBSTS[PS].

The periodic schedule is used to manage all isochronous and interrupt transfer streams. The base of the periodic schedule is the periodic frame list. Software links schedule data structures to the periodic frame list to produce a graph of scheduled data structures. The graph represents an appropriate sequence of transactions on the USB. Figure 16-47 illustrates isochronous transfers (using iTDs and siTDs) with a period of one are linked directly to the periodic frame list. Interrupt transfers (are managed with queue heads) and isochronous streams with periods other than one are linked following the period-one iTD/siTDs. Interrupt queue heads are linked into the frame list ordered by poll rate. Longer poll rates are

linked first (for example, closest to the periodic frame list), followed by shorter poll rates, with queue heads with a poll rate of one, on the very end.

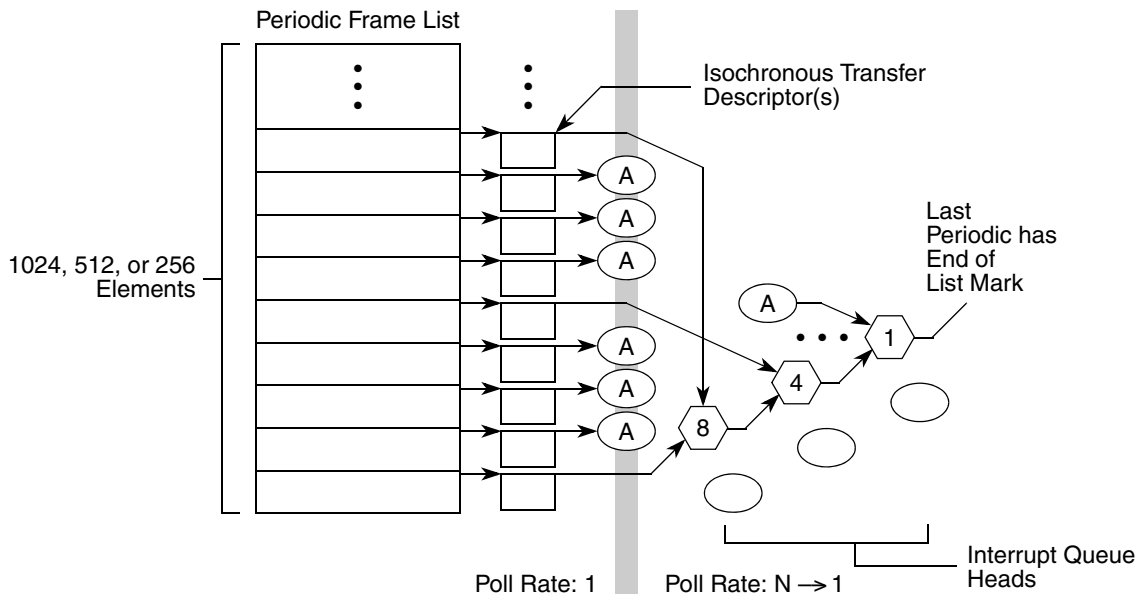


Figure 16-47. Example Periodic Schedule

16.6.8 Managing Isochronous Transfers Using iTDs

The structure of an iTD is presented in isochronous (high-speed) transfer descriptor (iTID). There are four distinct sections to an iTD:

- Next link pointer
 - This is the first field.
 - This field is for schedule linkage purposes only.
- Transaction description array
 - This is an eight-element array.
 - Each element represents control and status information for one microframe's worth of transactions for a single high-speed isochronous endpoint.
- Buffer page pointer array
 - This is a 7-element array of physical memory pointers to data buffers.
 - These are 4K aligned pointers to physical memory.
- Endpoint capabilities
 - This area utilizes the unused low-order 12 bits of the buffer page pointer array.
 - Its fields are used across all transactions executed for this iTD, including endpoint addressing, transfer direction, maximum packet size and high-bandwidth multiplier.

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 microframes worth of transactions. When the host controller fetches an iTD, it uses FRINDEX register bits 2–0 to index into the transaction description array. When the first iTD in the periodic list is traversed after periodic schedule is enabled, the value of FRINDEX[2:0] may be other than 0, so the first transaction issued by the controller may be any of the eight available active transactions. If the active bit in the Status field of the indexed transaction description is cleared, the host controller ignores the iTD and follows the Next pointer to the next schedule data structure.

When the indexed active bit is a one the host controller continues to parse the iTD. It stores the indexed transaction description and the general endpoint information (device address, endpoint number, maximum packet size, etc.). It also uses the Page Select (PG) field to index the buffer pointer array, storing the selected buffer pointer and the next sequential buffer pointer. For example, if PG field is a 0, then the host controller will store Page 0 and Page 1.

The host controller constructs a physical data buffer address by concatenating the current buffer pointer (as selected using the current transaction description's PG field) and the transaction description's Transaction Offset field. The host controller uses the endpoint addressing information and I/O-bit to execute a transaction to the appropriate endpoint. When the transaction is complete, the host controller clears the active bit and writes back any additional status information to the Status field in the currently selected transaction description.

The data buffer associated with the iTD must be virtually contiguous memory. Seven page pointers are provided to support eight high-bandwidth transactions regardless of the starting packet's offset alignment into the first page. A starting buffer pointer (physical memory address) is constructed by concatenating the page pointer (example: page 0 pointer) selected by the active transaction descriptions' PG (example value: 0b00) field with the transaction offset field. As the transaction moves data, the host controller must detect when an increment of the current buffer pointer will cross a page boundary. When this occurs the host controller simply replaces the current buffer pointer's page portion with the next page pointer (example: page 1 pointer) and continues to move data. The size of each bus transaction is determined by the value in the Maximum Packet Size field. An iTD supports high-bandwidth pipes via the Mult (multiplier) field. When the Mult field is 1, 2, or 3, the host controller executes the specified number of Maximum Packet sized bus transactions for the endpoint in the current microframe. In other words, the Mult field represents a transaction count for the endpoint in the current microframe. If the Mult field is zero, the operation of the host controller is undefined. The transfer description is used to service all transactions indicated by the Mult field.

For OUT transfers, the value of the Transaction n Length field represents the total bytes to be sent during the microframe. The Mult field must be set by software to be consistent with Transaction n Length and Maximum Packet Size. The host controller will send the bytes in Maximum Packet Sized portions. After each transaction, the host controller decrements 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 possible data. During each IN transaction, the host controller must use Maximum Packet Size to detect packet babble errors. The host controller keeps the sum of bytes received in the Transaction n Length field.

After all transactions for the endpoint have completed for the microframe, Transaction n Length contains the total bytes received. The following actions can occur:

- If the final value of Transaction n Length is less than the value of Maximum Packet Size, less data was received than was allowed for from the associated endpoint. This short packet condition does not set USBSTS[UI] (USB interrupt). The host controller does not detect this condition.
- If the device sends more than Transaction n Length or Maximum Packet Size bytes (whichever is less), the host controller sets the Babble Detected bit and clears the Active bit. Note, that the host controller does not update the iTD field Transaction n Length in this error scenario.
- If the Mult field is greater than one, the host controller automatically executes the value of Mult transactions. The host controller does not execute all Mult transactions in the following cases:
 - The endpoint is an OUT and Transaction n Length goes to zero before all the Mult transactions have executed (ran out of data).
 - The endpoint is an IN and the endpoint delivers a short packet, or an error occurs on a transaction before Mult transactions have been executed.

The end of microframe may occur before all of the transaction opportunities have been executed. When this happens, the transfer state of the transfer description is advanced to reflect the progress that was made; the result is written back to the iTD; and the host controller proceeds to processing the next microframe.

16.6.8.2 Software Operational Model for iTDs

A client buffer request to an isochronous endpoint may span 1 to N microframes. When N is larger than one, system software may have to use multiple iTDs to read or write data with the buffer (if N is larger than eight, it must use more than one iTD).

Figure 16-48 illustrates the simple model of how a client buffer is mapped by system software to the periodic schedule (that is, the periodic frame list and a set of iTDs).

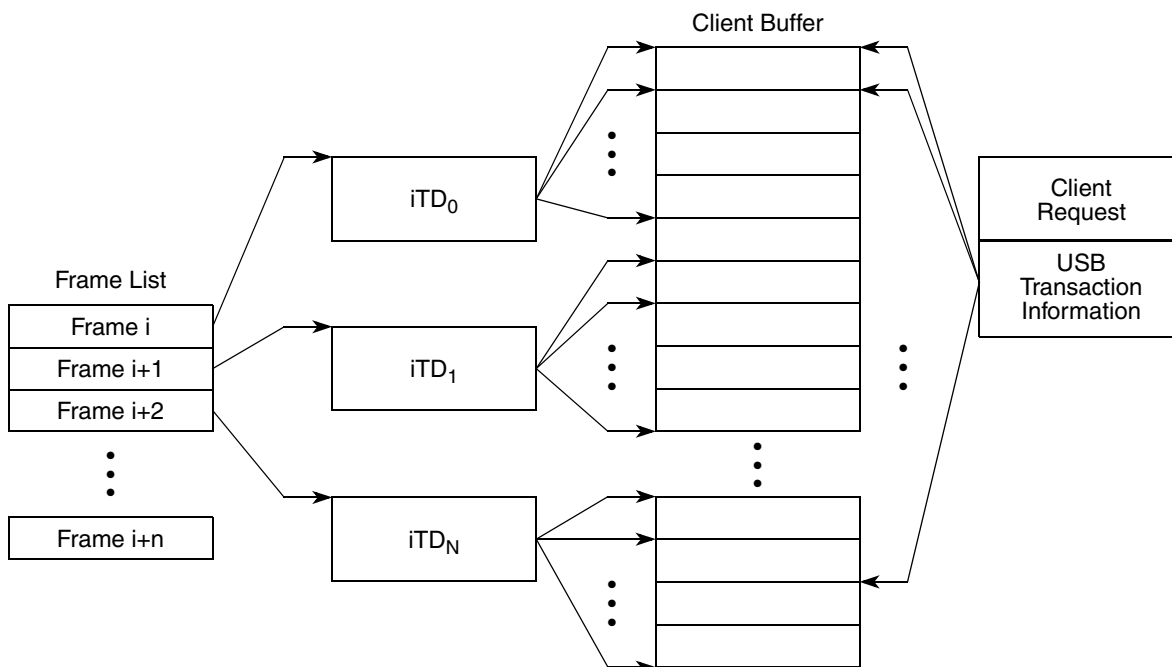


Figure 16-48. Example Association of iTDs to Client Request Buffer

On the right is the client description of its request. The description includes a buffer base address plus additional annotations to identify which portions of the buffer should be used with each bus transaction. In the middle is the iTD data structures used by the system software to service the client request. Each iTD can be initialized to service up to 24 transactions, organized into eight groups of up to three transactions each. Each group maps to one microframe's worth of transactions. The EHCI controller does not provide per transaction results within a microframe. It treats the per microframe transactions as a single logical transfer.

On the left is the host controller's frame list. System software establishes references from the appropriate locations in the frame list to each of the appropriate iTDs. If the buffer is large, system software can use a small set of iTDs to service the entire buffer. System software can activate the transaction description records (contained in each iTD) in any pattern required for the particular data stream.

As noted above, the client request includes a pointer to the base of the buffer and offsets into the buffer to annotate which buffer sections are to be used on each bus transaction that occurs on this endpoint. System software must initialize each transaction description in an iTD to ensure it uses the correct portion of the client buffer. For example, for each transaction description, the PG field is set to index the correct physical buffer page pointer and the Transaction Offset field is set relative to the correct buffer pointer page (for example, the same one referenced by the PG field). When the host controller executes a transaction it selects a transaction description record based on FRINDEX[2-0]. It then uses the current Page Buffer Pointer (as selected by the PG field) and concatenates to the transaction offset field. The result is a starting buffer address for the transaction. As the host controller moves data for the transaction, it must watch for a page wrap condition and properly advance to the next available Page Buffer Pointer. System software must not use the Page 6 buffer pointer in a transaction description where the length of the transfer will wrap

a page boundary. Doing so yields undefined behavior. The host controller hardware is not required to alias the page selector to page zero. USB 2.0 isochronous endpoints can specify a period greater than one. Software can achieve the appropriate scheduling by linking iTDs into the appropriate frames (relative to the frame list) and by setting appropriate transaction description elements active bits to a one.

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 microframes worth of transactions. The host controller is allowed to cache one (or more) of these data structures in order to reduce memory traffic. There are three basic caching models that account for the fact the isochronous data structures span 8 microframes. The three caching models are: no caching, microframe caching and frame caching.

When software is adding new isochronous transactions to the schedule, it always performs a read of the FRINDEX register to determine the current frame and microframe the host controller is currently executing. Of course, there is no information about where in the microframe the host controller is, so a constant uncertainty factor of one microframe has to be assumed. Combining the knowledge of where the host controller is executing with the knowledge of the caching model allows the definition of simple algorithms for how closely software can reliably work to the executing host controller.

No caching is indicated with a value of zero in the Isochronous Scheduling Threshold field. The host controller may pre-fetch data structures during a periodic schedule traversal (per microframe) but will always dump any accumulated schedule state at the end of the microframe. At the appropriate time relative to the beginning of every microframe, the host controller always begins schedule traversal from the frame list. Software can use the value of the FRINDEX register (plus the constant 1 uncertainty-factor) to determine the approximate position of the executing host controller. When no caching is selected, software can add an isochronous transaction as near as 2 microframes in front of the current executing position of the host controller.

Frame caching is indicated with a non-zero value in bit [7] of the Isochronous Scheduling Threshold field. In the frame-caching model, system software assumes that the host controller caches one (or more) isochronous data structures for an entire frame (8 microframes). Software uses the value of the FRINDEX register (plus the constant 1 uncertainty) to determine the current microframe/frame (assume modulo 8 arithmetic in adding the constant 1 to the microframe number). For any current frame N , if the current microframe is 0 to 6, then software can safely add isochronous transactions to Frame $N + 1$. If the current microframe is 7, then software can add isochronous transactions to Frame $N + 2$.

Microframe caching is indicated with a non-zero value in the least-significant 3 bits of the Isochronous Scheduling Threshold field. System software assumes the host controller caches one or more periodic data structures for the number of microframes indicated in the Isochronous Scheduling Threshold field. For example, if the count value were 2, then the host controller keeps a window of two microframes worth of

state (current microframe, plus the next) on chip. On each microframe boundary, the host controller releases the current microframe state and begins accumulating the next microframe state.

16.6.9 Asynchronous Schedule

The asynchronous schedule traversal is enabled or disabled through USBCMD[ASE] (asynchronous schedule enable). If USBCMD[ASE] is cleared, then the host controller simply does not try to access the asynchronous schedule via the ASYNCLISTADDR register. Likewise, if USBCMD[ASE] is set, the host controller does use the ASYNCLISTADDR register to traverse the asynchronous schedule. Modifications to USBCMD[ASE] are not necessarily immediate. Rather the new value of the bit will only be taken into consideration the next time the host controller needs to use the value of the ASYNCLISTADDR register to get the next queue head.

USBSTS[AS] indicates status of the asynchronous schedule. System software enables (or disables) the asynchronous schedule by writing a one (or zero) to USBCMD[ASE]. Software then can poll USBSTS[AS] to determine when the asynchronous schedule has made the desired transition. Software must not modify USBCMD[ASE] unless the value of USBCMD[ASE] equals that of the USBSTS[AS] (asynchronous schedule status).

The asynchronous schedule is used to manage all Control and Bulk transfers. Control and Bulk transfers are managed using queue head data structures. The asynchronous schedule is based at the ASYNCLISTADDR register. The default value of the ASYNCLISTADDR register after reset is undefined and the schedule is disabled when USBCMD[ASE] is cleared.

Software may only write this register with defined results when the schedule is disabled, for example, USBCMD[ASE] and the USBSTS[AS] are cleared. System software enables execution from the asynchronous schedule by writing a valid memory address (of a queue head) into this register. Then software enables the asynchronous schedule by setting USBCMD[ASE]. The asynchronous schedule is actually enabled when USBSTS[AS] is set.

When the host controller begins servicing the asynchronous schedule, it begins using the value of the ASYNCLISTADDR register. It reads the first referenced data structure and begins executing transactions and traversing the linked list as appropriate. When the host controller completes processing the asynchronous schedule, it retains the value of the last accessed queue head's horizontal pointer in the ASYNCLISTADDR register. Next time the asynchronous schedule is accessed, this is the first data structure that is serviced. This provides round-robin fairness for processing the asynchronous schedule.

A host controller completes processing the asynchronous schedule when one of the following events occur:

- The end of a microframe occurs.
- The host controller detects an empty list condition
- The schedule has been disabled through USBCMD[ASE].

The queue heads in the asynchronous list are linked into a simple circular list as shown in [Figure 16-44](#). Queue head data structures are the only valid data structures that may be linked into the asynchronous schedule. An isochronous transfer descriptor (iT_D or si_{TD}) 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 USBCMD[ASE] to a one.

When inserting a queue head into an active list, software must ensure that the schedule is always coherent from the host controllers' point of view. This means that the system software must ensure that all queue head pointer fields are valid. For example qTD pointers have T-Bits set or reference valid qTDs and the Horizontal Pointer references a valid queue head data structure. The following algorithm represents the functional requirements:

```

InsertQueueHead (pQHeadCurrent, pQueueHeadNew)
--
-- Requirement: all inputs must be properly initialized.
--
-- pQHeadCurrent is a pointer to a queue head that is
-- already in the active list
-- pQHeadNew is a pointer to the queue head to be added
--
-- This algorithm links a new queue head into a existing
-- list
--
pQueueHeadNew.HorizontalPointer = pQueueHeadCurrent.HorizontalPointer
pQueueHeadCurrent.HorizontalPointer = physicalAddressOf(pQueueHeadNew)
End InsertQueueHead

```

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 USBCMD[ASE] to a zero. Software can determine when the list is idle when USBSTS[AS] is cleared. The normal mode of operation is that software removes queue heads from the asynchronous schedule without shutting it down. Software must not remove an active queue head from the schedule. Software should first deactivate all active qTDs, wait for the queue head to go inactive, then remove the queue head from the asynchronous list. Software removes a queue head from the asynchronous list using the following algorithm. Software merely must ensure all of the link pointers reachable by the host controller are kept consistent.

```

UnlinkQueueHead (pQHeadPrevious, pQueueHeadToUnlink, pQHeadNext)
--
-- Requirement: all inputs must be properly initialized.
--

```

```

-- pQHeadPrevious is a pointer to a queue head that
-- references the queue head to remove
-- pQHeadToUnlink is a pointer to the queue head to be
-- removed
-- pQheadNext is a pointer to a queue head still in the
-- schedule. Software provides this pointer with the
-- following strict rules:
-- if the host software is one queue head, then
-- pQHeadNext must be the same as
-- QueueheadToUnlink.HorizontalPointer. If the host
-- software is unlinking a consecutive series of
-- queue heads, QHeadNext must be set by software to
-- the queue head remaining in the schedule.
--
-- This algorithm unlinks a queue head from a circular list
--
pQueueHeadPrevious.HorizontalPointer = pQueueHeadToUnlink.HorizontalPointer
pQueueHeadToUnlink.HorizontalPointer = pQHeadNext
End UnlinkQueueHead

```

If software removes the queue head with the H-bit set, it must select another queue head still linked into the schedule and set its H-bit. This should be completed before removing the queue head. The requirement is that software keep one queue head in the asynchronous schedule, with its H-bit set. At the point software has removed one or more queue heads from the asynchronous schedule, it is unknown whether the host controller has a cached pointer to them. Similarly, it is unknown how long the host controller might retain the cached information, as it is implementation dependent and may be affected by the actual dynamics of the schedule load. Therefore, once software has removed a queue head from the asynchronous list, it must retain the coherency of the queue head (link pointers). It cannot disturb the removed queue heads until it knows that the host controller does not have a local copy of a pointer to any of the removed data structures.

The method software uses to determine when it is safe to modify a removed queue head is to handshake with the host controller. The handshake mechanism allows software to remove items from the asynchronous schedule, then execute a simple, lightweight handshake that is used by software as a key that it can free (or reuse) the memory associated the data structures it has removed from the asynchronous schedule.

The handshake is implemented with three bits in the host controller. The first bit is a command bit (USBCMD[IAA]—interrupt on async advance doorbell) that allows software to inform the host controller that something has been removed from its asynchronous schedule. The second bit is a status bit (USBSTS[AAI]—interrupt on async advance) that the host controller sets after it has released all on-chip state that may potentially reference one of the data structures just removed. When the host controller sets this status bit, it also clears the command bit. The third bit is an interrupt enable (USBINTR[AAE]—interrupt on async advance enable) that is matched with the status bit. If the status bit is set and the interrupt enable bit is set, then the host controller asserts a hardware interrupt.

Figure 16-49 illustrates a general example where consecutive queue heads (B and C) are unlinked from the schedule using the algorithm above. Before the unlink operation, the host controller has a copy of queue head A.

The unlink algorithm requires that as software unlinks each queue head, the unlinked queue head is loaded with the address of a queue head that will remain in the asynchronous schedule.

When the host controller observes that doorbell bit being set, it makes a note of the local reachable schedule information. In this example, the local reachable schedule information includes both queue heads (A & B). It is sufficient that the host controller can set the status bit (and clear the doorbell bit) as soon as it has traversed beyond current reachable schedule information (that is, traversed beyond queue head (B) in this example).

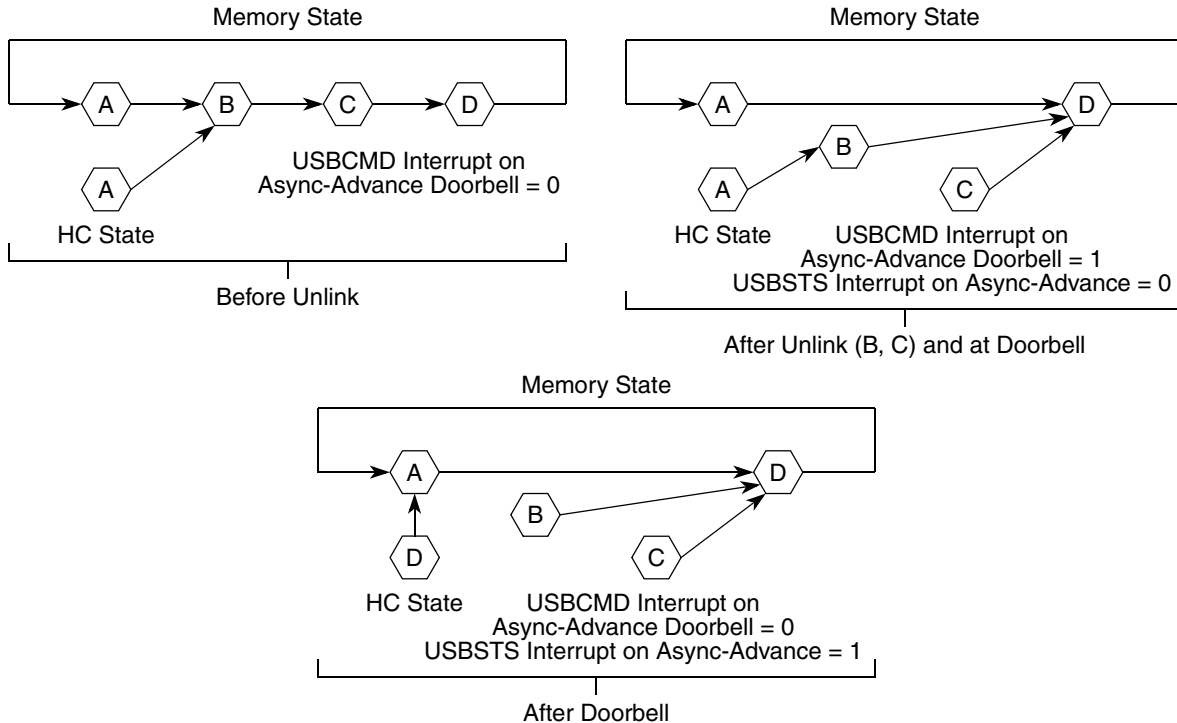


Figure 16-49. Generic Queue Head Unlink Scenario

Alternatively, a host controller implementation is allowed to traverse the entire asynchronous schedule list (for example, observed the head of the queue (twice)) before setting USBSTS[AAI].

Software may re-use the memory associated with the removed queue heads after it observes USBSTS[AAI] is set, following assertion of the doorbell. Software should acknowledge the interrupt on async advance status as indicated in the USBSTS register, before using the doorbell handshake again

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 16-41](#)) defines an H-bit in the queue head, which allows software to mark a queue head as being the head of the reclaim list. host controller also keeps a 1-bit flag in the USBSTS register (Reclamation) that is cleared when the host controller observes a queue head with the H-bit set. The reclamation flag in the status register is set when any USB transaction from the asynchronous schedule is executed (or whenever the asynchronous schedule starts, see [Section 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.

Figure 16-50 shows an example illustrating the H-bit in a schedule.

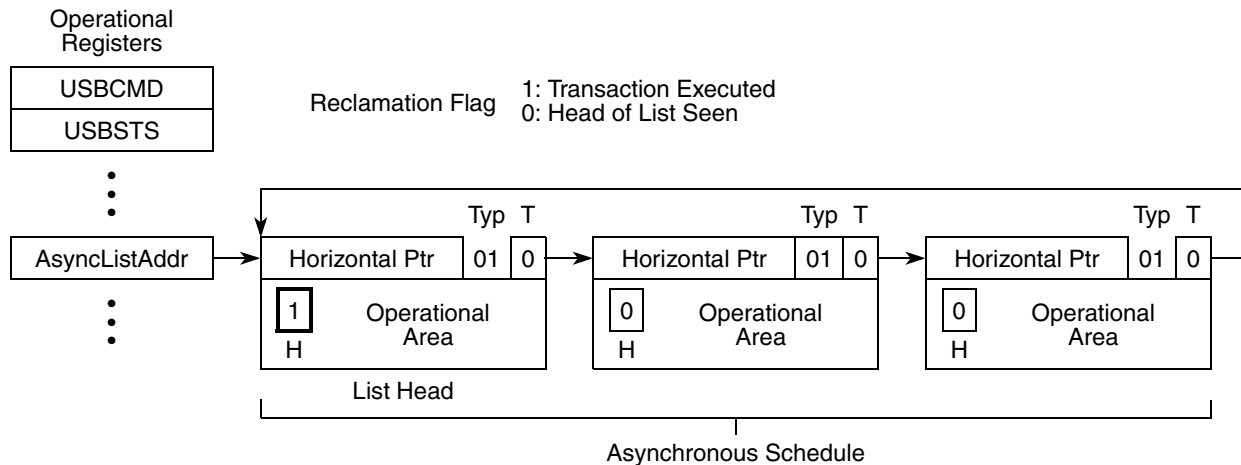


Figure 16-50. Asynchronous Schedule List with Annotation to Mark Head of List

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 microframe. In addition, it may have idled itself early in a microframe. When this occurs (idles early in the microframe) the host controller must occasionally reactivate during the microframe and traverse the asynchronous schedule to determine whether any progress can be made. Asynchronous schedule Start Events are defined to be:

- Whenever the host controller transitions from the periodic schedule to the asynchronous schedule. If the periodic schedule is disabled and the asynchronous schedule is enabled, then the beginning of the microframe is equivalent to the transition from the periodic schedule, or
- The asynchronous schedule traversal restarts from a sleeping state.

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 (RCL) in the USBSTS register. The host controller tests for an empty schedule just after it fetches a new queue head while traversing the asynchronous schedule. The host controller sets USBSTS[RCL] whenever an asynchronous schedule traversal Start Event occurs. USBSTS[RCL] is also set whenever the host controller executes a transaction while traversing the asynchronous schedule. The host controller clears USBSTS[RCL] whenever it finds a queue head with its H-bit set. Software should only set a queue head's H-bit if the queue head is in the asynchronous schedule. If software sets the H-bit in an interrupt queue head, the resulting behavior is undefined. The host controller may clear USBSTS[RCL] when executing from the periodic schedule.

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 16-51 illustrates these requirements.

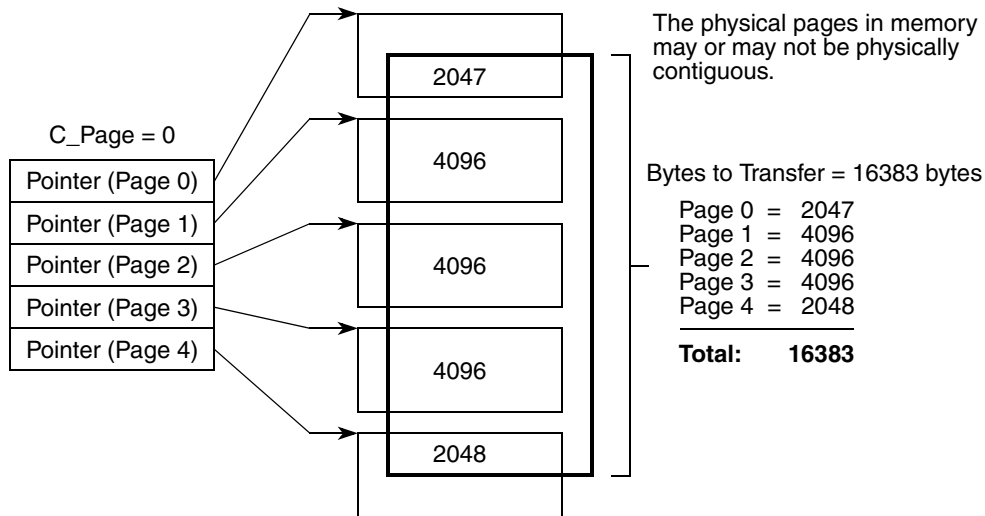


Figure 16-51. Example Mapping of qTD Buffer Pointers to Buffer Pages

The buffer pointer list in the qTD is long enough to support a maximum transfer size of 20K bytes. This case occurs when all five buffer pointers are used and the first offset is zero. A qTD handles a 16Kbyte buffer with any starting buffer alignment.

The host controller uses the C_Page field as an index value to determine which buffer pointer in the list should be used to start the current transaction. The host controller uses a different buffer pointer for each physical page of the buffer. This is always true, even if the buffer is physically contiguous.

The host controller must detect when the current transaction spans a page boundary and automatically move to the next available buffer pointer in the page pointer list. The next available pointer is reached by incrementing C_Page and pulling the next page pointer from the list. Software must ensure there are sufficient buffer pointers to move the amount of data specified in the Bytes to Transfer field.

Figure 16-51 illustrates a nominal example of how System software would initialize the buffer pointers list and the C_Page field for a transfer size of 16383 bytes. C_Page is cleared. The upper 20-bits of Page 0 references the start of the physical page. Current Offset (the lower 12-bits of queue head Dword 7) holds the offset in the page for example, 2049 (for example, 4096-2047). The remaining page pointers are set to reference the beginning of each subsequent 4K page.

For the first transaction on the qTD (assuming a 512-byte transaction), the host controller uses the first buffer pointer (page 0 because C_Page is cleared) and concatenates the Current Offset field. The 512 bytes are moved during the transaction, the Current Offset and Total Bytes to Transfer are adjusted by 512 and written back to the queue head working area.

During the 4th transaction, the host controller needs 511 bytes in page 0 and one byte in page 1. The host controller will increment C_Page (to 1) and use the page 1 pointer to move the final byte of the transaction. After the 4th transaction, the active page pointer is the page 1 pointer and Current Offset has rolled to one, and both are written back to the overlay area. The transactions continue for the rest of the buffer, with the

host controller automatically moving to the next page pointer (that is, C_Page) when necessary. There are three conditions for how the host controller handles C_Page.

- The current transaction does not span a page boundary. The value of C_Page is not adjusted by the host controller.
- The current transaction does span a page boundary. The host controller must detect the page cross condition and advance to the next buffer while streaming data to/from the USB.
- The current transaction completes on a page boundary (that is, the last byte moved for the current transaction is the last byte in the page for the current page pointer). The host controller must increment C_Page before writing back status for the transaction.

Note that the only valid adjustment the host controller may make to C_Page is to increment by one.

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 microframe within a 1 millisecond period a transaction should be executed for the queue head. Software must ensure that all queue heads in the periodic schedule have S-Mask set to a non-zero value. An S-mask with a zero value in the context of the periodic schedule yields undefined results.

If the desired poll rate is greater than one frame, system software can use a combination of queue head linking and S-Mask values to spread interrupts of equal poll rates through the schedule so that the periodic bandwidth is allocated and managed in the most efficient manner possible. Some examples are illustrated in [Table 16-65](#).

Table 16-65. 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 microframes) is linked into the periodic schedule so that it is reachable from the periodic frame list locations indicated in the previous column. In addition, the S-Mask field in the queue head is set to 0x01, indicating that the transaction for the endpoint should be executed on the bus during microframe 0 of the frame.
0, 2, 4, 6, 8, ... S-Mask = 0x02	Another example of a queue head with a 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 microframe 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 16-54](#)). 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 16-66](#) illustrates the state transition table for the host controller's responsibility for maintaining the PING protocol. Refer to Chapter 8 in the *USB Specification, Revision 2.0* for detailed description on the Ping protocol.

Table 16-66. Ping Control State Transition Table

Current	Event		Next
	Host	Device	
Do Ping	PING	Nak	Do Ping
Do Ping	PING	Ack	Do OUT
Do Ping	PING	XactErr 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 16-54](#). The defined ping protocol allows the host to be imprecise on the initialization of the ping protocol (that is, start in Do OUT when there is no information whether there is space on the device or not). The host controller manages the Ping State bit. System software sets the initial value in the queue head when it initializes a queue head. The host controller preserves the Ping State bit across all queue advancements. This means that when a new qTD is written into the queue head overlay area, the previous value of the Ping State bit is preserved.

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.

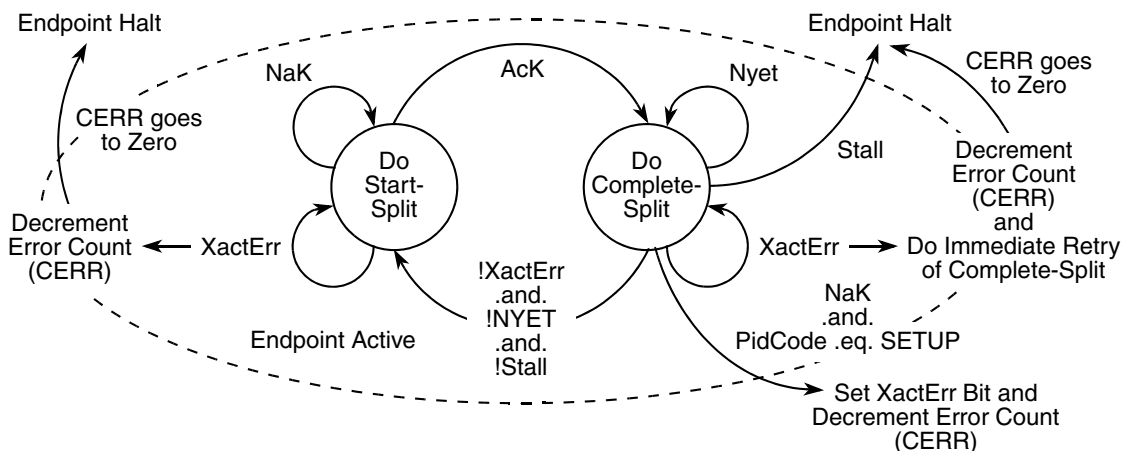


Figure 16-52. Host Controller Asynchronous Schedule Split-Transaction State Machine

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 microframe to execute the retry, the host controller ensures that the next time the host controller begins executing from the Asynchronous schedule, it must begin executing from this queue head. If another start-split (for some other endpoint) is sent to the transaction translator before the complete-split is really completed, the transaction translator could dump the results (which were never delivered to the host). This is why the core specification states the retries must be immediate. When the host controller returns to the asynchronous schedule in the next microframe, the first transaction from the schedule will be the retry for this endpoint. If Cerr went to zero, the host controller halts the queue.
- NAK. The target endpoint Nak'd the full- or low-speed transaction. The state of the transfer is not advanced and the state is exited. If the PID Code is a SETUP, then the Nak response is a protocol error. The XactErr status bit is set and the Cerr field is decremented.
- STALL. The target endpoint responded with a STALL handshake. The host controller sets the halt bit in the status byte, retires the qTD but does not attempt to advance the queue.

If the PID Code indicates an IN, then any of following responses are expected:

- **DATA0/1.** On reception of data, the host controller ensures the PID matches the expected data toggle and checks CRC. If the packet is good, the host controller advances the state of the transfer (for example, moves the data pointer by the number of bytes received, decrements the BytesToTransfer field by the number of bytes received, and toggles the dt bit). The host controller then exits this state. The response and advancement of transfer may trigger other processing events, such as retirement of the qTD and advancement of the queue.

If the data sequence PID does not match the expected, the data is ignored, the transfer state is not advanced and this state is exited.

If the PID Code indicates an OUT/SETUP, then any of following responses are expected:

- **ACK.** The target endpoint accepted the data, so the host controller must advance the state of the transfer. The Current Offset field is incremented by Maximum Packet Length or Bytes to Transfer, whichever is less. The Bytes To Transfer field is decremented by the same amount and the data toggle bit (dt) is toggled. The host controller then exits this state.

Advancing the transfer state may cause other processing events such as retirement of the qTD and advancement of the queue.

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 microframes the start-splits and complete-splits for each endpoint will occur. The characteristics of the transaction translator are such that the high-speed transaction protocol must execute during explicit microframes, or the data or response information in the pipeline is lost. [Figure 16-53](#) illustrates the general scheduling boundary conditions that are supported by the EHCI periodic schedule

and queue head data structure. The S and C_n labels indicate microframes where software can schedule start-splits and complete splits (respectively).

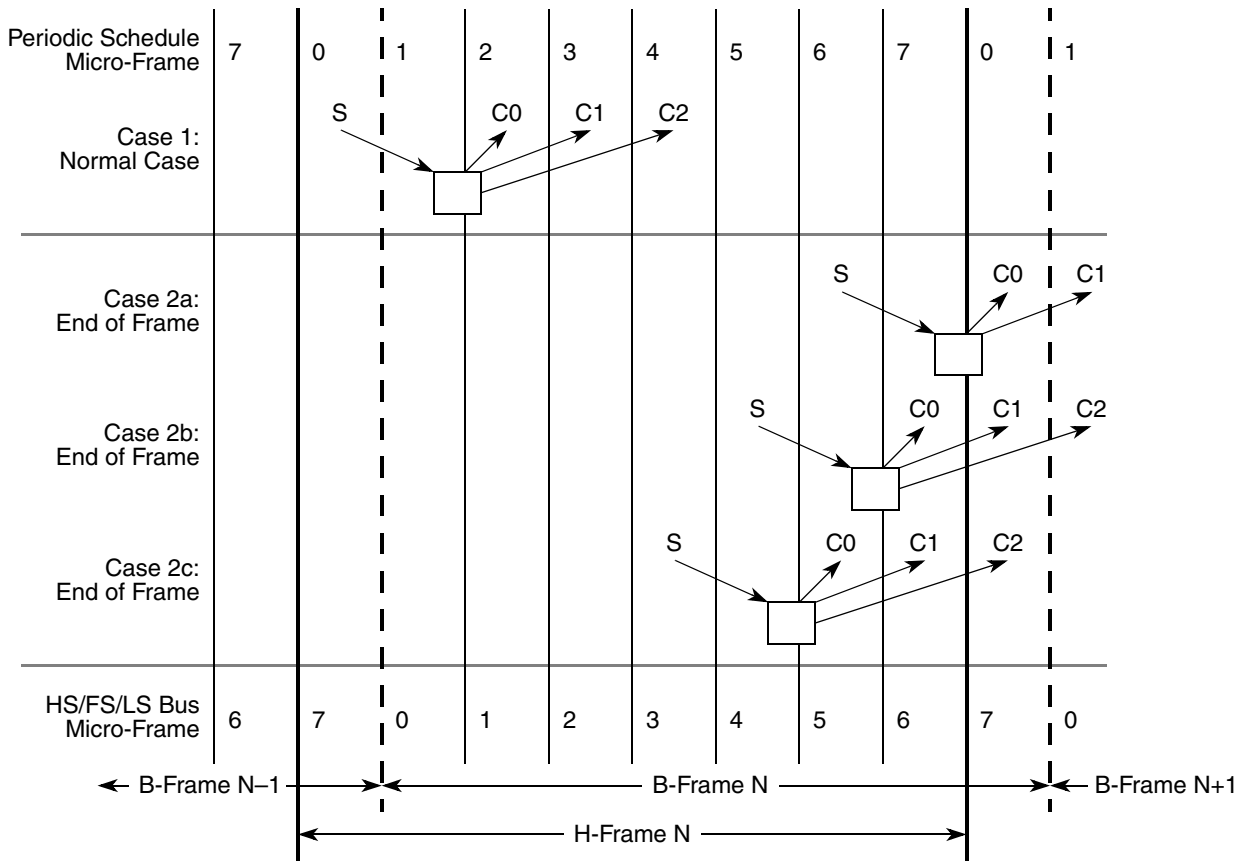


Figure 16-53. Split Transaction, Interrupt Scheduling Boundary Conditions

The scheduling cases are:

- Case 1: The normal scheduling case is where the entire split transaction is completely bounded by a frame (H-Frame in this case).
- Case 2a through Case 2c: The USB 2.0 hub pipeline rules states clearly, when and how many complete-splits must be scheduled to account for earliest to latest execution on the full/low-speed link. The complete-splits may span the H-Frame boundary when the start-split is in microframe 4 or later. When this occurs, the H-Frame to B-Frame alignment requires that the queue head be reachable from consecutive periodic frame list locations. System software cannot build an efficient schedule that satisfies this requirement unless it uses FSTNs. [Figure 16-54](#) illustrates the general layout of the periodic schedule.

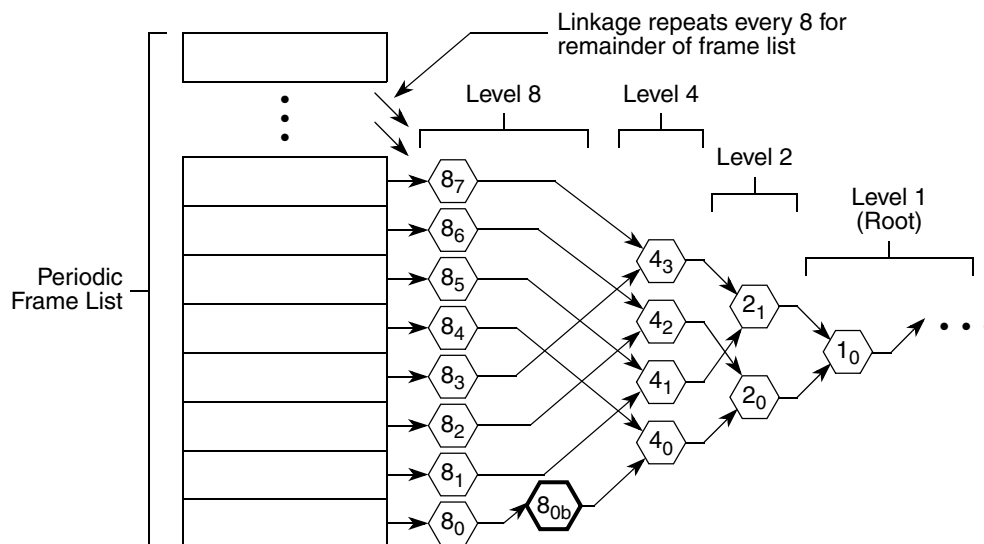


Figure 16-54. General Structure of EHCI Periodic Schedule Utilizing Interrupt Spreading

The periodic frame list is effectively the leaf level a binary tree, which is always traversed leaf to root. Each level in the tree corresponds to a 2^N poll rate. Software can efficiently manage periodic bandwidth on the USB by spreading interrupt queue heads that have the same poll rate requirement across all the available paths from the frame list. For example, system software can schedule eight poll rate 8 queue heads and account for them once in the high-speed bus bandwidth allocation.

When an endpoint is allocated an execution footprint that spans a frame boundary, the queue head for the endpoint must be reachable from consecutive locations in the frame list. An example would be if 8_{0b} were such an endpoint. Without additional support on the interface, to get 8_{0b} reachable at the correct time, software would have to link 8_1 to 8_{0b} . It would then have to move 4_1 and everything linked after into the same path as 4_0 . This upsets the integrity of the binary tree and disallows the use of the spreading technique.

FSTN data structures are used to preserve the integrity of the binary-tree structure and enable the use of the spreading technique. [Section 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 16-54](#)). This bit is used to track the current state of the split transaction.
- **Frame S-mask.** This is a bit-field where-in system software sets a bit corresponding to the microframe (within an H-Frame) that the host controller should execute a start-split transaction. This is always qualified by the value of the SplitXState bit in the Status field of the queue head. For example, referring to [Figure 16-53](#), case one, the S-mask would have a value of `0b0000_0001` indicating that if the queue head is traversed by the host controller, and the SplitXState indicates Do_Start, and the current microframe as indicated by `FRINDEX[2-0]` is 0, then execute a start-split transaction.

- Frame C-mask. This is a bit-field where system software sets one or more bits corresponding to the microframes (within an H-Frame) that the host controller should execute complete-split transactions. The interpretation of this field is always qualified by the value of the SplitXState bit in the Status field of the queue head. For example, referring to [Figure 16-53](#), case one, the C-mask would have a value of 0b0001_1100 indicating that if the queue head is traversed by the host controller, and the SplitXState indicates Do_Complete, and the current microframe as indicated by FRINDEX[2-0] is 2, 3, or 4, then execute a complete-split transaction. It is software's responsibility to ensure that the translation between H-Frames and B-Frames is correctly performed when setting bits in S-mask and C-mask.

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 microframes 2 through 7 it simply follows the node's Normal Path Link Pointer to access the next schedule data structure. Note that the FSTN's Normal Path Link Pointer[T] bit may set, which the host controller must interpret as the end of periodic list mark.

When the host controller encounters a Save-Place FSTN in microframes 0 or 1, it saves the value of the Normal Path Link Pointer and sets an internal flag indicating that it is executing in Recovery Path mode. Recovery Path mode modifies the host controller's rules for how it traverses the schedule and limits which data structures are considered for execution of bus transactions. The host controller continues executing in Recovery Path mode until it encounters a Restore FSTN or it determines that it has reached the end of the microframe.

The rules for schedule traversal and limited execution while in Recovery Path mode are:

- Always follow the Normal Path Link Pointer when it encounters an FSTN that is a Save-Place indicator. The host controller must not recursively follow Save-Place FSTNs. Therefore, while executing in Recovery Path mode, it must never follow an FSTN's Back Path Link Pointer.
- Do not process an siTD or iTD data structure; simply follow its Next Link Pointer.
- Do not process a QH (Queue Head) whose EPS field indicates a high-speed device; simply follow its Horizontal Link Pointer.
- When a QH's EPS field indicates a Full/Low-speed device, the host controller only considers it for execution if its SplitXState is DoComplete (note: this applies whether the PID Code indicates an

IN or an OUT). Refer to the *EHCI Specification* for a complete list of additional conditions that must be met in general for the host controller to issue a bus transaction. Note that the host controller must not execute a Start-split transaction while executing in Recovery Path mode. Refer to the *EHCI Specification* for special handling when in Recovery Path mode.

- Stop traversing the recovery path when it encounters an FSTN that is a Restore indicator. The host controller unconditionally uses the saved value of the Save-Place FSTN's Normal Path Link Pointer when returning to the normal path traversal. The host controller must clear the context of executing a Recovery Path when it restores schedule traversal to the Save-Place FSTN's Normal Path Link Pointer.

If the host controller determines that there is not enough time left in the microframe to complete processing of the periodic schedule, it abandons traversal of the recovery path, and clears the context of executing a recovery path. The result is that at the start of the next consecutive microframe, the host controller starts traversal at the frame list.

An example traversal of a periodic schedule that includes FSTNs is illustrated in [Figure 16-55](#).

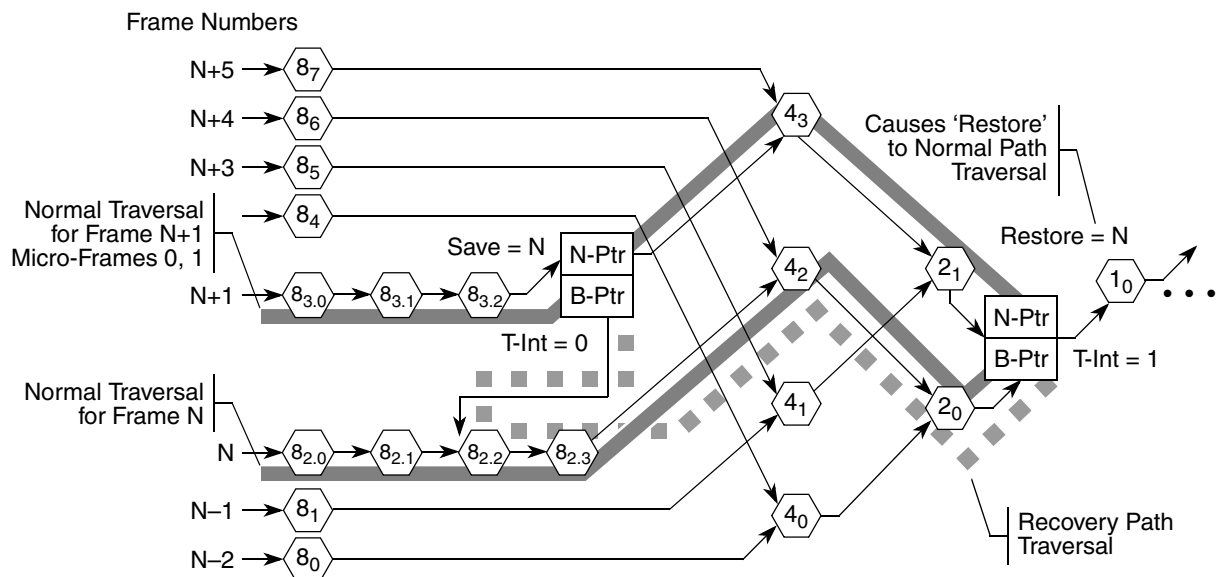


Figure 16-55. Example Host Controller Traversal of Recovery Path via FSTNs

In frame N (microframes 0–7), for this example, the host controller traverses all of the schedule data structures utilizing the Normal Path Link Pointers in any FSTNs it encounters. This is because the host controller has not yet encountered a Save-Place FSTN so it is not executing in Recovery Path mode. When it encounters the Restore FSTN, (Restore-N), during microframes 0 and 1, it uses Restore-N. Normal Path Link Pointer to traverse to the next data structure (that is, normal schedule traversal). This is because the host controller must use a Restore FSTN's Normal Path Link Pointer when not executing in a Recovery-Path mode. The nodes traversed during frame N include: $\{8_{2,0}, 8_{2,1}, 8_{2,2}, 8_{2,3}, 4_2, 2_0, \text{Restore-N}, 1_0 \dots\}$.

In frame N+1 (microframes 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 16-55 with a large dashed line). The host controller continues traversing data structures on the recovery path and executing only those bus transactions as noted above, on the recovery path until it reaches Restore FSTN (Restore-N). Restore-N.Back Path Link Pointer.T-bit is set (definition of a Restore indicator), so the host controller exits Recovery Path mode by clearing the internal Recovery Path mode flag and commences (restores) schedule traversal using the saved value of the Save-Place FSTN's Normal Path Link Pointer (for example, Save-N.Normal Path Link Pointer). The nodes traversed during these microframes include: { $8_{3,0}$, $8_{3,1}$, $8_{3,2}$, Save-A, $8_{2,2}$, $8_{2,3}$, 4_2 , 2_0 , Restore-N, 4_3 , 2_1 , Restore-N, 1_0 ...}.

In frame N+1 (microframes 2-7), when the host controller encounters Save-Path FSTN Save-N, it unconditionally follows Save-N.Normal Path Link Pointer. The nodes traversed during these microframes include: { $8_{3,0}$, $8_{3,1}$, $8_{3,2}$, 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 is used to mark the end of the periodic list.
- When the schedule does have elements linked at a poll rate level of one, a Restore FSTN must be the first data structure on the poll rate one list. All traversal paths from the frame list converge on the poll-rate one list. System software must ensure that Recovery Path mode is exited before the host controller is allowed to traverse the poll rate level one list.
- A Save-Place FSTN's Back Path Link Pointer must reference a queue head data structure. The referenced queue head must be reachable from the previous frame list location. In other words, if the Save-Place FSTN is reachable from frame list offset N, then the FSTN's Back Path Link Pointer must reference a queue head that is reachable from frame list offset N-1.

Software should make the schedule as efficient as possible. What this means in this context is that software should have no more than one Save-Place FSTN reachable in any single frame. Note there are times when two (or more, depending on the implementation) could exist as full-/low-speed footprints change with bandwidth adjustments. This could occur, for example when a bandwidth rebalance causes system software to move the Save-Place FSTN from one poll rate level to another. During the transition, software must preserve the integrity of the previous schedule until the new schedule is in place.

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 microframe number in which the complete-split was executed. The host controller always checks C-prog-mask before executing a complete-split transaction. If the previous complete-splits have not been executed then it means one (or more) have been skipped and data has potentially been lost.
- **FrameTag.** This field is used by the host controller during the complete-split portion of the split transaction to tag the queue head with the frame number (H-Frame number) when the next complete split must be executed.
- **S-bytes.** This field can be used to store the number of data payload bytes sent during the start-split (if the transaction was an OUT). The S-bytes field must be used to accumulate the data payload bytes received during the complete-splits (for an IN).

16.6.12.2.5 Split Transaction Execution State Machine for Interrupt

In the following section, all references to microframe are in the context of a microframe within an H-Frame.

As with asynchronous full- and low-speed endpoints, a split-transaction state machine is used to manage the split transaction sequence. Aside from the fields defined in the queue head for scheduling and tracking the split transaction, the host controller calculates one internal mechanism that is also used to manage the split transaction. The internal calculated mechanism is:

- **cMicroFrameBit.** This is a single-bit encoding of the current microframe number. It is an eight-bit value calculated by the host controller at the beginning of every microframe. It is calculated from the three least significant bits of the FRINDEX register (that is, $cMicroFrameBit = (1 \text{ shifted-left}(FRINDEX[2-0]))$). The cMicroFrameBit has at most one bit asserted, which always

corresponds to the current microframe number. For example, if the current microframe is 0, then `cMicroFrameBit` will equal `0b0000_0001`.

The variable `cMicroFrameBit` is used to compare against the S-mask and C-mask fields to determine whether the queue head is marked for a start- or complete-split transaction for the current microframe.

Figure 16-56 illustrates how a complete interrupt split transaction is managed. There are two phases to each split transaction. The first is a single start-split transaction, which occurs when the `SplitXState` is at `Do_Start` and the single bit in `cMicroFrameBit` has a corresponding bit active in `QH[S-mask]`. The transaction translator does not acknowledge the receipt of the periodic start-split, so the host controller unconditionally transitions the state to `Do_Complete`. Due to the available jitter in the transaction translator pipeline, there is more than one complete-split transaction scheduled by software for the `Do_Complete` state. This translates simply to the fact that there are multiple bits set in the `QH[C-mask]` field.

The host controller keeps the queue head in the `Do_Complete` state until the split transaction is complete (see definition below), or an error condition triggers the three-strikes-rule (for example, after the host tries the same transaction three times, and each encounters an error, the host controller stops retrying the bus transaction and halts the endpoint, thus requiring system software to detect the condition and perform system-dependent recovery).

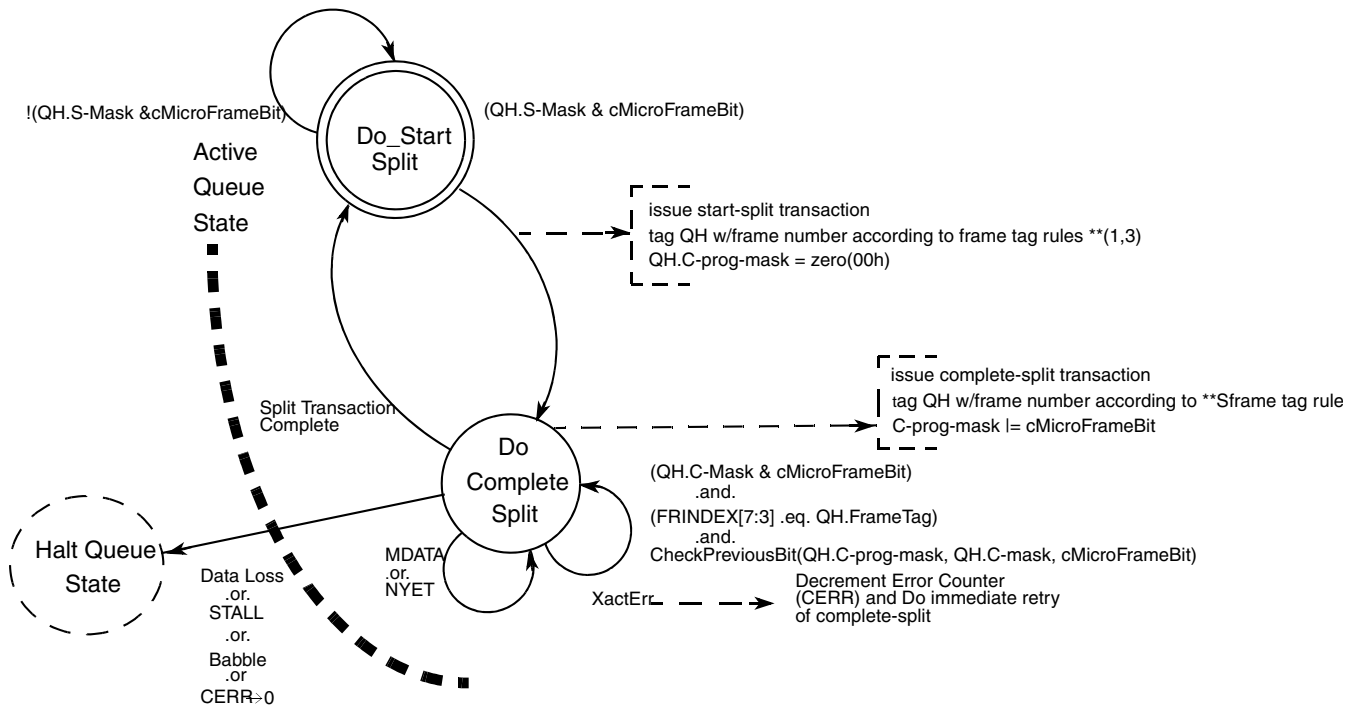


Figure 16-56. Split Transaction State Machine for Interrupt

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 16.6.12.2.7, “Periodic Interrupt—Do-Complete-Split,”](#) for the definition of 'Last'.

Each time the host controller visits a queue head in this state (once within the Execute Transaction state), bit-wise ANDs QH[S-mask] with cMicroFrameBit to determine whether to execute a start-split. If the result is non-zero, then the host controller issues a start-split transaction. If the PID Code field indicates an IN transaction, the host controller must zero-out the QH[S-bytes] field. After the split-transaction has been executed, the host controller sets up state in the queue head to track the progress of the complete-split phase of the split transaction. Specifically, it records the expected frame number into QH[FrameTag] field, sets C-prog-mask to zero (0x00), and exits this state. Note that the host controller must not adjust the value of Cerr as a result of completion of a start-split transaction.

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 microframe.
- **Test B.** QH[FrameTag] is compared with the current contents of FRINDEX[7–3]. An equal indicates a match.
- **Test C.** The complete-split progress bit vector is checked to determine whether the previous bit is set, indicating that the previous complete-split was appropriately executed. An example algorithm for this test is provided below:

```

Algorithm Boolean CheckPreviousBit(QH.C-prog-mask, QH.C-mask, cMicroFrameBit)
Begin
-- Return values:
-- TRUE - no error
-- FALSE - error
--
Boolean rvalue = TRUE;
previousBit = cMicroframeBit logical-rotate-right(1)
-- Bit-wise anding previousBit with C-mask indicates
-- whether there was an intent
    
```

```

-- to send a complete split in the previous microframe. So,
-- if the
-- 'previous bit' is set in C-mask, check C-prog-mask to
-- make sure it
-- happened.
If (previousBit bitAND QH.C-mask)then
    If not(previousBit bitAND QH.C-prog-mask) then
        rvalue = FALSE;
    End if
End If
-- If the C-prog-mask already has a one in this bit position,
-- then an aliasing
-- error has occurred. It will probably get caught by the
-- FrameTag Test, but
-- at any rate it is an error condition that as detectable here
-- should not allow
-- a transaction to be executed.
If (cMicroFrameBit bitAND QH.C-prog-mask) then
    rvalue = FALSE;
End if
return (rvalue)
End Algorithm

```

- Test D. Check to see if a start-split should be executed in this microframe. Note this is the same test performed in the Do Start Split state. Whenever it evaluates to TRUE and the controller is NOT processing in the context of a Recovery Path mode, it means a start-split should occur in this microframe. Test D and Test A evaluating to TRUE at the same time is a system software error. Behavior is undefined.

If (A .and. B .and. C .and. not(D)) then the host controller will execute a complete-split transaction. When the host controller commits to executing the complete-split transaction, it updates QH[C-prog-mask] by bit-ORing with cMicroFrameBit. On completion of the complete-split transaction, the host controller records the result of the transaction in the queue head and sets QH[FrameTag] to the expected H-Frame number. The effect to the state of the queue head and thus the state of the transfer depends on the response by the transaction translator to the complete-split transaction. The following responses have the effects (note that any responses that result in decrementing of the Cerr will result in the queue head being halted by the host controller if the result of the decrement is zero):

- NYET (and Last)

On each NYET response, the host controller checks to determine whether this is the last complete-split for this split transaction. Last is defined in this context as the condition where all of the scheduled complete-splits have been executed. If it is the last complete-split (with a NYET response), then the transfer state of the queue head is not advanced (never received any data) and this state exited. The transaction translator must have responded to all the complete-splits with NYETs, meaning that the start-split issued by the host controller was not received. The start-split should be retried at the next poll period.

The test for whether this is the Last complete split can be performed by XOR QH[C-mask] with QH[C-prog-mask]. If the result is all zeros then all complete-splits have been executed. When this condition occurs, the XactErr status bit is set and the Cerr field is decremented.
- NYET (and not Last)

See above description for testing for Last. The complete-split transaction received a NYET response from the transaction translator. Do not update any transfer state (except for C-prog-mask and FrameTag) and stay in this state. The host controller must not adjust Cerr on this response.

- **Transaction Error (XactErr).** Timeout, data CRC failure, etc.
The Cerr field is decremented and the XactErr bit in the Status field is set. The complete split transaction is immediately retried (if Cerr is non-zero). If there is not enough time in the microframe to complete the retry and the endpoint is an IN, or Cerr is decremented to a zero from a one, the queue is halted. If there is not enough time in the microframe to complete the retry and the endpoint is an OUT and Cerr is not zero, then this state is exited (that is, return to Do Start Split). This results in a retry of the entire OUT split transaction, at the next poll period. Refer to Chapter 11 Hubs (specifically the section on full- and low-speed interrupts) in the *USB Specification Revision 2.0* for detailed requirements on why these errors must be immediately retried.
- **ACK**
This can only occur if the target endpoint is an OUT. The target endpoint ACK'd the data and this response is a propagation of the endpoint ACK up to the host controller. The host controller must advance the state of the transfer. The Current Offset field is incremented by Maximum Packet Length or Bytes to Transfer, whichever is less. The field Bytes To Transfer is decremented by the same amount. And the data toggle bit (dt) is toggled. The host controller will then exit this state for this queue head. The host controller must reload Cerr with maximum value on this response. Advancing the transfer state may cause other process events such as retirement of the qTD and advancement of the queue.
- **MDATA**
This response will only occur for an IN endpoint. The transaction translator responded with zero or more bytes of data and an MDATA PID. The incremental number of bytes received is accumulated in QH[S-bytes]. The host controller must not adjust Cerr on this response.
- **DATA0/1**
This response may only occur for an IN endpoint. The number of bytes received is added to the accumulated byte count in QH[S-bytes]. The state of the transfer is advanced by the result and the host controller exits this state for this queue head.
Advancing the transfer state may cause other processing events such as retirement of the qTD and advancement of the queue.
If the data sequence PID does not match the expected, the entirety of the data received in this split transaction is ignored, the transfer state is not advanced and this state is exited.
- **NAK**
The target endpoint Nak'd the full- or low-speed transaction. The state of the transfer is not advanced, and this state is exited. The host controller must reload Cerr with maximum value on this response.
- **ERR**
There was an error during the full- or low-speed transaction. The ERR status bit is set, Cerr is decremented, the state of the transfer is not advanced, and this state is exited.
- **STALL**

The queue is halted (an exit condition of the Execute Transaction state). The status field bits: Active bit is cleared and the Halted bit is set and the qTD is retired. Responses which are not enumerated in the list or which are received out of sequence are illegal and may result in undefined host controller behavior. The other possible combinations of tests A, B, C, and D may indicate that data or response was lost. [Table 16-67](#) lists the possible combinations and the appropriate action.

Table 16-67. Interrupt IN/OUT Do Complete Split State Execution Criteria

Condition	Action	Description
not(A) not(D)	Ignore QHD	Neither a start nor complete-split is scheduled for the current microframe. Host controller should continue walking the schedule.
A not(C)	If PIDCode = IN Halt QHD If PIDCode = OUT Retry start-split	Progress bit check failed. This means a complete-split has been missed. There is the possibility of lost data. If PID Code is an IN, then the Queue head must be halted. If PID Code is an OUT, then the transfer state is not advanced and the state exited (for example, start-split is retried). This is a host-induced error and does not effect Cerr. In either case, set the Missed Microframe bit in the status field to a one.
A not(B) C	If PIDCode = IN Halt QHD If PIDCode = OUT Retry start-split	QH.FrameTag test failed. This means that exactly one or more H-Frames have been skipped. This means complete-splits and have missed. There is the possibility of lost data. If PID Code is an IN, then the Queue head must be halted. If PID Code is an OUT, then the transfer state is not advanced and the state exited (for example, start-split is retried). This is a host-induced error and does not effect Cerr. In either case, set the Missed Microframe bit in the status field to a one.
A B C not(D)	Execute complete-split	This is the non-error case where the host controller executes a complete-split transaction.
D	If PIDCode = IN Halt QHD If PIDCode = OUT Retry start-split	This is a degenerate case where the start-split was issued, but all of the complete-splits were skipped and all possible intervening opportunities to detect the missed data failed to fire. If PID Code is an IN, then the Queue head must be halted. If PID Code is an OUT, then the transfer state is not advanced and the state exited (for example, start-split is retried). This is a host-induced error and does not effect Cerr. In either case, set the Missed Microframe bit in the status field to a one. Note that when executing in the context of a Recovery Path mode, the host controller is allowed to process the queue head and take the actions indicated above, or it may wait until the queue head is visited in the normal processing mode. Regardless, the host controller must not execute a start-split in the context of a executing in a Recovery Path mode.

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 16-53](#)).
- Rule 2: If the current value of FRINDEX[2–0] is 7, QH[FrameTag] is set to $FRINDEX[7–3] + 1$. This accommodates staying in Do Complete Split for cases 2a, 2b, and 2c in [Figure 16-53](#).

- Rule 3: If transitioning from Do_Start Split to Do Complete Split and the current value of FRINDEX[2–0] is not 6, or currently in Do Complete Split and the current value of (FRINDEX[2–0]) is not 7, FrameTag is set to FRINDEX[7–3]. This accommodates all other cases in [Figure 16-53](#).

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 microframe, the host controller must not issue the start-split bus transaction; it must clear the Active bit.

System software must save transfer state before setting the I-bit. This is required so that it can correctly determine what transfer progress (if any) occurred after the I-bit was set and the host controller executed its final bus-transaction and cleared the Active bit.

After system software has updated the S-mask and C-mask, it must then reactivate the queue head. Since the Active bit and the I-bit cannot be updated with the same write, system software needs to use the following algorithm to coherently re-activate a queue head that has been stopped using the I-bit.

1. Set the Halted bit, then
2. Clear the I-bit, then
3. Set the Active bit and clear the Halted bit in the same write.

Setting the Halted bit inhibits the host controller from attempting to advance the queue between the time the I-bit is cleared and the Active bit is set.

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 microframes the start-splits and complete-splits for each full-speed isochronous endpoint occur. The requirements described in [Section 16.6.12.2.1, “Split Transaction Scheduling Mechanisms for Interrupt,”](#) apply.

[Figure 16-57](#) illustrates the general scheduling boundary conditions that are supported by the EHCI periodic schedule. The S_n and C_n labels indicate microframes where software can schedule start- and complete-splits (respectively). The H-Frame boundaries are marked with a large, solid bold vertical line.

The B-Frame boundaries are marked with a large, bold, dashed line. The bottom of Figure 16-57 illustrates the relationship of an siTD to the H-Frame.

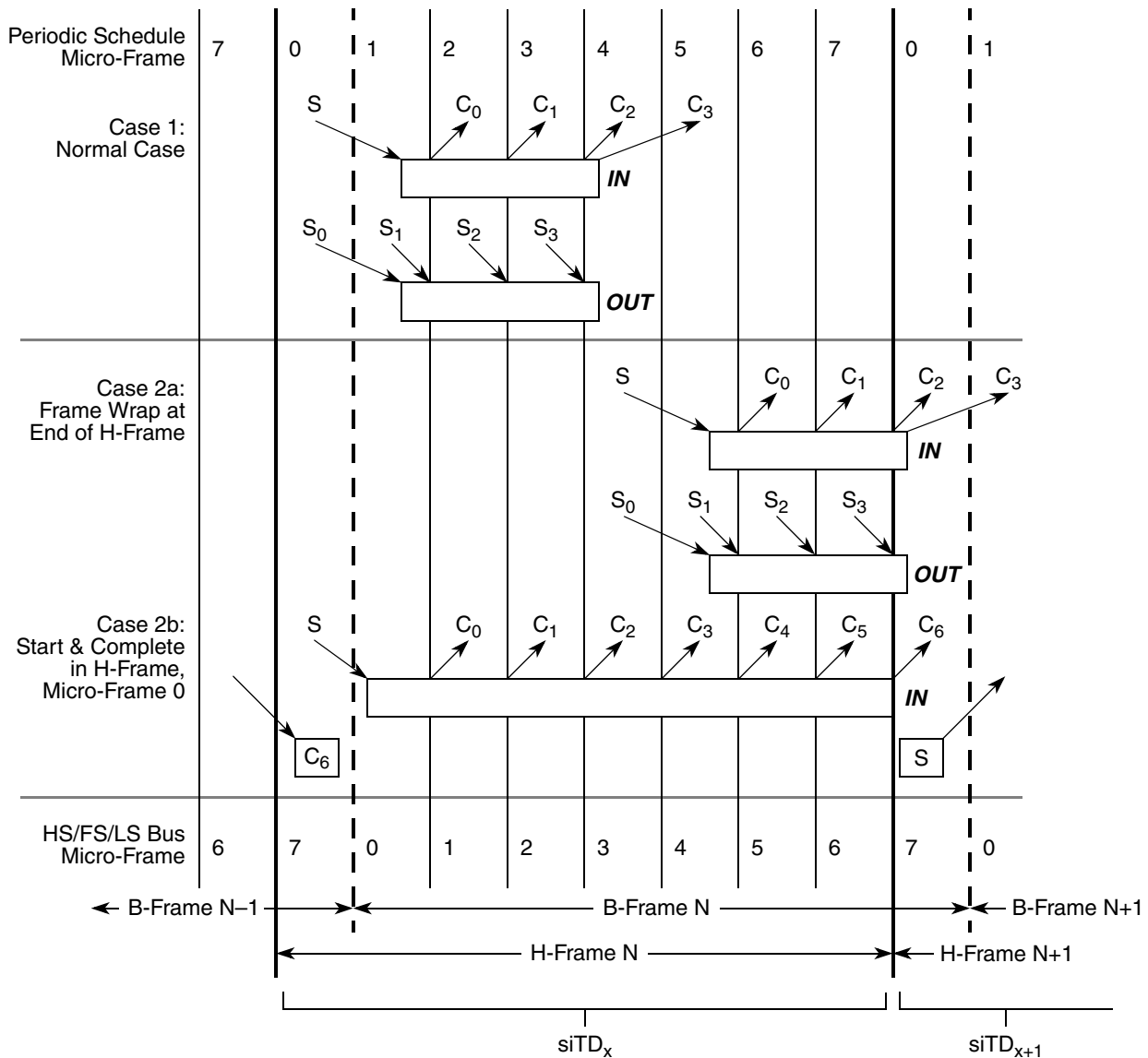


Figure 16-57. Split Transaction, Isochronous Scheduling Boundary Conditions

When the endpoint is an isochronous OUT, there are only start-splits, and no complete-splits. When the endpoint is an isochronous IN, there is at most one start-split and one to N complete-splits. The scheduling boundary cases are:

- Case 1: The entire split transaction is completely bounded by an H-Frame. For example, the start-splits and complete-splits are all scheduled to occur in the same H-Frame.
- Case 2a: This boundary case is where one or more (at most two) complete-splits of a split transaction IN are scheduled across an H-Frame boundary. This can only occur when the split transaction has the possibility of moving data in B-Frame, microframes 6 or 7 (H-Frame microframe 7 or 0). When an H-Frame boundary wrap condition occurs, the scheduling of the split

transaction spans more than one location in the periodic list.(for example, it takes two siTDs in adjacent periodic frame list locations to fully describe the scheduling for the split transaction).

Although the scheduling of the split transaction may take two data structures, all of the complete-splits for each full-speed IN isochronous transaction must use only one data pointer. For this reason, siTDs contain a back pointer.

Software must never schedule full-speed isochronous OUTs across an H-Frame boundary.

- Case 2b: This case can only occur for a very large isochronous IN. It is the only allowed scenario where a start-split and complete-split for the same endpoint can occur in the same microframe. Software must enforce this rule by scheduling the large transaction first. Large is defined to be anything larger than 579 byte maximum packet size.

A subset of the same mechanisms employed by full- and low-speed interrupt queue heads are employed in siTDs to schedule and track the portions of isochronous split transactions. The following fields are initialized by system software to instruct the host controller when to execute portions of the split transaction protocol:

- SplitXState
This is a single bit residing in the Status field of an siTD (see [Table 16-48](#)). 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 microframe (within an H-Frame) that the host controller should execute a start-split transaction. This is always qualified by the value of the SplitXState bit. For example, referring to the IN example in [Figure 16-57](#), case 1, the S-mask would have a value of 0b0000_0001 indicating that if the siTD is traversed by the host controller, and the SplitXState indicates Do Start Split, and the current microframe as indicated by FRINDEX[2–0] is 0, then execute a start-split transaction.
- Frame C-mask
This is a bit-field where system software sets one or more bits corresponding to the microframes (within an H-Frame) that the host controller should execute complete-split transactions. The interpretation of this field is always qualified by the value of the SplitXState bit. For example, referring to the IN example in [Figure 16-57](#), case 1, the C-mask would have a value of 0b 0011_1100 indicating that if the siTD is traversed by the host controller, and the SplitXState indicates Do Complete Split, and the current microframe as indicated by FRINDEX[2–0] is 2, 3, 4, or 5, then execute a complete-split transaction.
- Back Pointer
This field in a siTD is used to complete an IN split-transaction using the previous H-Frame's siTD. This is only used when the scheduling of the complete-splits span an H-Frame boundary.

There exists a one-to-one relationship between a high-speed isochronous split transaction (including all start- and complete-splits) and one full-speed isochronous transaction. An siTD contains (amongst other things) buffer state and split transaction scheduling information. An siTD's buffer state always maps to one full-speed isochronous data payload. This means that for any full-speed transaction payload, a single siTD's data buffer must be used. This rule applies to both IN 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 16-58 illustrates some examples. On the top are examples of the full-speed transaction footprints for the boundary scheduling cases described above. In the middle are time-frame references for both the B-Frames (HS/FS/LS Bus) and the H-Frames. On the bottom is illustrated the relationship between the scope of an siTD description and the time references. Each H-Frame corresponds to a single location in the periodic frame list. The implication is that each siTD is reachable from a single periodic frame list location at a time.

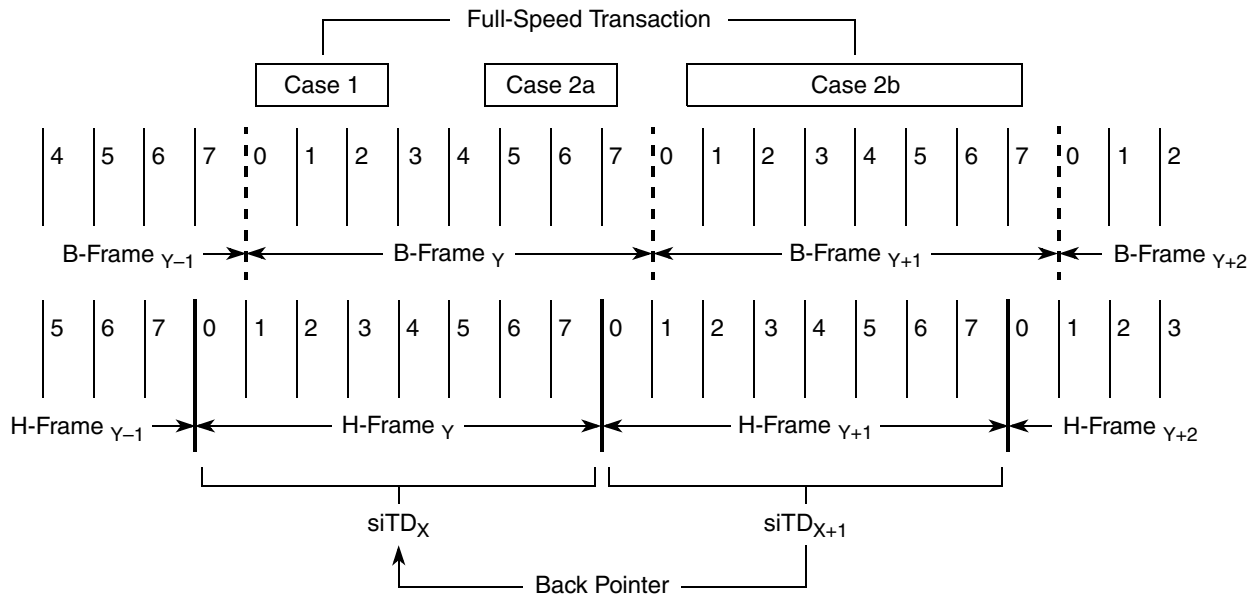


Figure 16-58. siTD Scheduling Boundary Examples

Each case is described below:

- Case 1: One siTD is sufficient to describe and complete the isochronous split transaction because the whole isochronous split transaction is tightly contained within a single H-Frame.
- Case 2a, 2b: Although both INs and OUTs can have these footprints, OUTs always take only one siTD to schedule. However, INs (for these boundary cases) require two siTDs to complete the scheduling of the isochronous split transaction. siTD_X is used to always issue the start-split and the first N complete-splits. The full-speed transaction (for these cases) can deliver data on the full-speed bus segment during microframe 7 of H-Frame_{Y+1}, or microframe 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, microframe 1. This is mandated as a rule so that case 2a and case 2b can be discriminated. According to the core USB specification, the long isochronous transaction illustrated in Case 2b, could be scheduled so that the start-split was in microframe 1 of H-Frame N and the last complete-split would need to occur in microframe 1 of H-Frame N+1. However, it is impossible to discriminate between cases 2a and case 2b, which has significant impact on the complexity of the host controller.

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 microframes), timeouts and corrupted data received.

In similar kind to interrupt split-transactions, the portions of the split transaction protocol must execute in the microframes they are scheduled. The queue head data structure used to manage full- and low-speed interrupt has several mechanisms for tracking when portions of a transaction have occurred. Isochronous transfers use siTDs for their transfers and the data structures are only reachable using the schedule in the exact microframe in which they are required (so all the mechanism employed for tracking in queue heads is not required for siTDs). Software has the option of reusing siTD several times in the complete periodic schedule. However, it must ensure that the results of split transaction N are consumed and the siTD re-initialized (activated) before the host controller gets back to the siTD (in a future microframe).

Split-transaction isochronous OUTs utilize a low-level protocol to indicate which portions of the split transaction data have arrived. Control over the low-level protocol is exposed in an siTD using the fields Transaction Position (TP) and Transaction Count (T-count). If the entire data payload for the OUT split transaction is larger than 188 bytes, there will be more than one start-split transaction, each of which require proper annotation. If host hold-offs occur, then the sequence of annotations received from the host will not be complete, which is detected and handled by the transaction translator. See [Section 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 microframe (FRINDEX[2-0]) number in which the complete-split was executed. The host controller always checks C-prog-mask before executing a complete-split transaction. If the previous complete-splits have not been executed, then it means one (or more) have been skipped and data has potentially been lost. System software is required to initialize this field to zero before setting an siTD's Active bit to a one.

If a transaction translator returns with the final data before all of the complete-splits have been executed, the state of the transfer is advanced so that the remaining complete-splits are not executed. It is important to note that an IN siTD is retired based solely on the responses from the transaction translator to the complete-split transactions. This means, for example, that it is possible for a transaction translator to respond to a complete-split with an MDATA PID. The number of bytes in the MDATA's data payload could cause the siTD[Total Bytes to Transfer] field to decrement to zero. This response can occur, before all of the scheduled complete-splits have been executed. In other interface, data structures (for example, high-speed data streams through queue heads), the transition of Total Bytes to Transfer to zero signals the end of the transfer and results in clearing the Active bit. However, in this case, the result has not been delivered by the transaction translator and the host must continue with the next complete-split transaction to extract the residual transaction state. This scenario occurs because of the pipeline rules for a transaction translator. In summary, the periodic pipeline rules require that on a microframe boundary, the transaction translator holds the final two bytes received (if it has not seen an End Of Packet (EOP)) in the full-speed bus pipe stage and gives the remaining bytes to the high-speed pipeline stage. At the microframe boundary, the transaction translator could have received the entire packet (including both CRC bytes) but not received the packet EOP. In the next microframe, the transaction translator responds with an MDATA and sends all of the data bytes (with the two CRC bytes being held in the full-speed pipeline stage). This could cause the siTD to decrement its Total Bytes to Transfer field to zero, indicating it has received all expected data. The host must still execute one more (scheduled) complete-split transaction in order to extract the results of the full-speed transaction from the transaction translator (for example, the transaction translator may have detected a CRC failure, and this result must be forwarded to the host).

If the host experiences hold-offs that cause the host controller to skip one or more (but not all) scheduled split transactions for an isochronous OUT, then the protocol to the transaction translator is not consistent and the transaction translator detects and reacts to the problem. Likewise, for host hold-offs that cause the host controller to skip one or more (but not all) scheduled split transactions for an isochronous IN, the C-prog-mask is used by the host controller to detect errors. However, if the host experiences a hold-off that causes it to skip all of an siTD, or an siTD expires during a host hold off (for example, a hold-off occurs and the siTD is no longer reachable by the host controller in order for it to report the hold-off event), then system software must detect that the siTDs have not been processed by the host controller (for example, state not advanced) and report the appropriate error to the client driver.

16.6.12.3.3 Split Transaction Execution State Machine for Isochronous

In this section, all references to microframe are in the context of a microframe within an H-Frame.

If the Active bit in the Status byte is a zero, the host controller ignores the siTD and continues traversing the periodic schedule. Otherwise the host controller processes the siTD as specified below. A split

transaction state machine is used to manage the split-transaction protocol sequence. The host controller uses the fields defined in Section 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 16-59 illustrates the state machine for managing an `siTD` through an isochronous split transaction. Bold, dotted circles denote the state of the Active bit in the Status field of a `siTD`. The Bold, dotted arcs denote the transitions between these states. Solid circles denote the states of the split transaction state machine and the solid arcs denote the transitions between these states. Dotted arcs and boxes reference actions that take place either as a result of a transition or from being in a state.

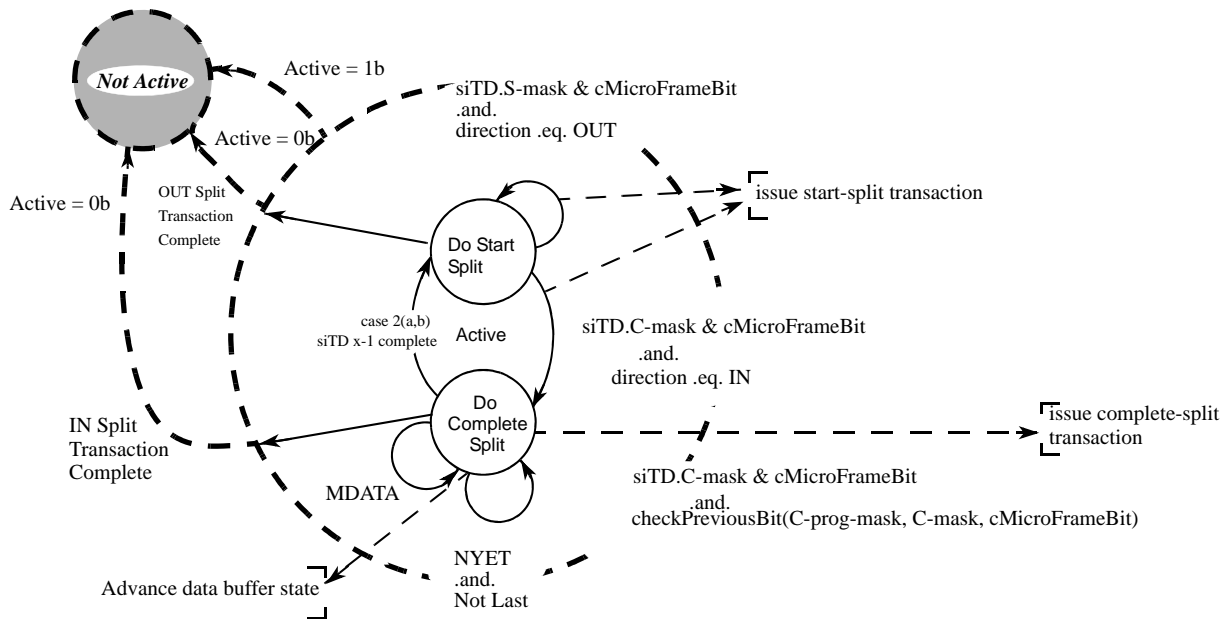


Figure 16-59. Split Transaction State Machine for Isochronous

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 16-58](#)) is used to determine the initial value of TP. The initial cases are summarized in [Table 16-68](#).

Table 16-68. Initial Conditions for OUT siTD TP and T-Count Fields

Case	T-Count	TP	Description
1, 2a	=1	ALL	When the OUT data payload is less than (or equal to) 188 bytes, only one start-split is required to move the data. The one start-split must be marked with an ALL.
1, 2a	!=1	BEGIN	When the OUT data payload is greater than 188 bytes more than one start-split must be used to move the data. The initial start-split must be marked with a BEGIN.

After each start-split transaction is complete, the host controller updates T-count and TP appropriately so that the next start-split is correctly annotated. [Table 16-69](#) illustrates all of the TP and T-count transitions, which must be accomplished by the host controller.

Table 16-69. Transaction Position (TP)/Transaction Count (T-Count) Transition Table

TP	T-Count Next	TP Next	Description
ALL	0	N/A	Transition from ALL, to done.
BEGIN	1	END	Transition from BEGIN to END. Occurs when T-count starts at 2.
BEGIN	!=1	MID	Transition from BEGIN to MID. Occurs when T-count starts at greater than 2.
MID	!=1	MID	TP stays at MID while T-count is not equal to 1 (for example, greater than 1). This case can occur for any of the scheduling boundary cases where the T-count starts greater than 3.
MID	1	END	Transition from MID to END. This case can occur for any of the scheduling boundary cases where the T-count starts greater than 2.

The start-split transactions do not receive a handshake from the transaction translator, so the host controller always advances the transfer state in the siTD after the bus transaction is complete. To advance the transfer state the following operations take place:

- The siTD[Total Bytes To Transfer] and the siTD[Current Offset] fields are adjusted to reflect the number of bytes transferred.
- The siTD[P] (page select) bit is updated appropriately.

- The siTD[TP] and siTD[T-count] fields are updated appropriately as defined in [Table 16-69](#).

These fields are then written back to the memory based siTD. The S-mask is fixed for the life of the current budget. As mentioned above, TP and T-count are set specifically in each siTD to reflect the data to be sent from this siTD. Therefore, regardless of the value of S-mask, the actual number of start-split transactions depends on T-count (or equivalently, Total Bytes to Transfer). The host controller must clear the Active bit when it detects that all of the schedule data has been sent to the bus. The host controller must clear the Active bit when it detects that all of the schedule data has been sent to the bus. Setting the Active bit to zero depends on siTD[TP] being 00 or 11, and siTD[Total Bytes to Transfer] decrements to 0. Software must ensure that TP, T-count and Total Bytes to Transfer are set to deliver the appropriate number of bus transactions from each siTD. An inconsistent combination yields undefined behavior.

If the host experiences hold-offs that cause the host controller to skip start-split transactions for an OUT transfer, the state of the transfer does not progress appropriately. The transaction translator observes protocol violations in the arrival of the start-splits for the OUT endpoint (that is, the transaction position annotation is incorrect as received by the transaction translator).

Example scenarios are described in [Section 16.6.12.3.7, “Split Transaction for Isochronous—Processing Example.”](#)

The host controller tracks 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 microframes. It can then clear the siTD's Active bit and stop execution of this siTD. This saves on both memory and high-speed bus bandwidth.

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 sequence in which they are applied depends on which microframe the host controller is currently executing, which means that the tests might not be applied until after the siTD referenced from the back pointer has been fetched. The individual tests are as follows.

- Test A
cMicroFrameBit is bit-wise ANDed with the siTD[C-mask] field. A non-zero result indicates that software scheduled a complete-split for this endpoint, during this microframe. This test is always applied to a newly fetched siTD that is in this state.
- Test B

The siTD[C-prog-mask] bit vector is checked to determine whether the previous complete splits have been executed. An example algorithm is given below (this is slightly different than the algorithm used in [Section 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, this is a case 2a or 2b scheduling boundary (see [Figure 16-57](#)). 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, the host controller executes a complete-split transaction using the transfer state of the current siTD. When the host controller commits to executing the complete-split transaction, it updates QH[C-prog-mask] by bit-ORing with cMicroFrameBit. The transfer state is advanced based on the completion status of the complete-split transaction. To advance the transfer state of an IN siTD, the host controller must perform the following actions:

1. Decrement the number of bytes received from siTD[Total Bytes To Transfer]
2. Adjust siTD[Current Offset] by the number of bytes received
3. Adjust the siTD[P] (page select) field if the transfer caused the host controller to use the next page pointer
4. Set any appropriate bits in the siTD[Status] field, depending on the results of the transaction.

Note that if the host controller encounters a condition where siTD[Total Bytes To Transfer] is zero, and it receives more data, the host controller must not write the additional data to memory. The siTD[Status-Active] bit must be cleared and the siTD[Status-Babble Detected] bit must be set. The fields siTD[Total Bytes To Transfer], siTD[Current Offset], and siTD[P] are not required to be updated as a result of this transaction attempt.

The host controller accepts (assuming good data packet CRC and sufficient room in the buffer as indicated by the value of siTD[Total Bytes To Transfer]) MDATA and DATA0/1 data payloads up to and including 192 bytes. The host controller may optionally clear siTD[Status-Active] and set siTD[Status-Babble Detected] when it receives MDATA or DATA0/1 with a data payload of more than 192 bytes. The following responses have the noted effects:

- **ERR**
The full-speed transaction completed with a time-out or bad CRC and this is a reflection of that error to the host. The host controller sets the ERR bit in the siTD[Status] field and clears the Active bit.
- **Transaction Error (XactErr)**
The complete-split transaction encounters a Timeout, CRC16 failure, etc. The siTD[Status] field XactErr field is set and the complete-split transaction must be retried immediately. The host controller must use an internal error counter to count the number of retries as a counter field is not

provided in the siTD data structure. The host controller will not retry more than two times. If the host controller exhausts the retries or the end of the microframe occurs, the Active bit is cleared.

- DATA_x (0 or 1)
This response signals that the final data for the split transaction has arrived. The transfer state of the siTD is advanced and the Active bit is cleared. If the Bytes To Transfer field has not decremented to zero (including the reception of the data payload in the DATA_x response), then less data than was expected, or allowed for was actually received. This short packet event does not set the USB interrupt status bit (USBSTS[UI]) to a one. The host controller will not detect this condition.
- NYET (and Last)
On each NYET response, the host controller also checks to determine whether this is the last complete-split for this split transaction. Last was defined in [Section 16.6.12.2.7, “Periodic Interrupt—Do-Complete-Split.”](#) If it is the last complete-split (with a NYET response), then the transfer state of the siTD is not advanced (never received any data) and the Active bit is cleared. No bits are set in the Status field because this is essentially a skipped transaction. The transaction translator must have responded to all the scheduled complete-splits with NYETs, meaning that the start-split issued by the host controller was not received. This result should be interpreted by system software as if the transaction was completely skipped. The test for whether this is the last complete split can be performed by XORing C-mask with C-prog-mask. A zero result indicates that all complete-splits have been executed.
- MDATA (and Last)
See above description for testing for Last. This can only occur when there is an error condition. Either there has been a babble condition on the full-speed link, which delayed the completion of the full-speed transaction, or software set up the S-mask and/or C-masks incorrectly. The host controller must set the XactErr bit and clear the Active bit.
- NYET (and not Last)
See above description for testing for Last. The complete-split transaction received a NYET response from the transaction translator. Do not update any transfer state (except for C-prog-mask) and stay in this state.
- MDATA (and not Last)
The transaction translator responds with an MDATA when it has partial data for the split transaction. For example, the full-speed transaction data payload spans from microframe X to X+1 and during microframe X, the transaction translator responds with an MDATA and the data accumulated up to the end of microframe X. The host controller advances the transfer state to reflect the number of bytes received.

If Test A succeeds, but Test B fails, it means that one or more of the complete-splits have been skipped. The host controller sets the Missed Micro-Frame status bit and clears the Active bit.

16.6.12.3.6 Complete-Split for Scheduling Boundary Cases 2a, 2b

Boundary cases 2a and 2b (INs only) (see [Figure 16-57](#)) require that the host controller use the transaction state context of the previous siTD to finish the split transaction. [Table 16-70](#) enumerates the transaction state fields.

Table 16-70. 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 siTD_X[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 16-70) 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 Example

There is an important difference between how the hardware/software manages the isochronous split transaction state machine and how it manages the asynchronous and interrupt split transaction state machines. The asynchronous and interrupt split transaction state machines are encapsulated within a single queue head. The progress of the data stream depends on the progress of each split transaction. In some respects, the split-transaction state machine is sequenced using the Execute Transaction queue head traversal state machine.

Isochronous is a pure time-oriented transaction/data stream. The interface data structures are optimized to efficiently describe transactions that need to occur at specific times. The isochronous split-transaction state machine must be managed across these time-oriented data structures. This means that system software must correctly describe the scheduling of split-transactions across more than one data structure.

Then the host controller must make the appropriate state transitions at the appropriate times, in the correct data structures.

For example, [Table 16-71](#) illustrates a few frames worth of scheduling required to schedule a case 2a full-speed isochronous data stream.

Table 16-71. Example Case 2a—Software Scheduling siTDs for an IN Endpoint

siTD _x		Micro-Frames								InitialSplitXState
#	Masks	0	1	2	3	4	5	6	7	
X	S-Mask					1				Do Start Split
	C-Mask	1	1					1	1	
X+1	S-Mask					1				Do Complete Split
	C-Mask	1	1					1	1	
X+2	S-Mask					1				Do Complete Split
	C-Mask	1	1					1	1	
X+3	S-Mask	Repeats previous pattern								Do Complete Split
	C-Mask									

This example shows the first three 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 microframe 4) and C-mask value of 0xC3 (one-bits in microframes 0,1, 6 and 7). Additionally, software ensures that the Back Pointer field of each siTD references the appropriate siTD data structure (and the Back Pointer T-bits are cleared).

The initial SplitXState of the first siTD is Do Start Split. The host controller will visit the first siTD eight times during frame X. The C-mask bits in microframes 0 and 1 are ignored because the state is Do Start Split. During microframe 4, the host controller determines that it can run a start-split (and does) and changes SplitXState to Do Complete Split. During microframes 6 and 7, the host controller executes complete-splits. Notice the siTD for frame X+1 has 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 microframes 0 and 1 it will detect that it must execute complete-splits.

During H-Frame X+1, microframe 0, the host controller detects that siTD_{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 microframe 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, microframe 1.

During H-Frame X+2, microframe 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 its next pointer without any state change updates to siTD_{X+2}.

During H-Frame X+2, microframe 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 microframe 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, assuming the CF-bit in the CONFIGFLAG register is set, is as follows:

1. Disable the periodic and asynchronous schedules by clearing the USBCMD[ASE] and USBCMD[PSE].
2. Place all enabled root ports into the suspended state by setting the Suspend bit in the PORTSC register (PORTSC[SUSP]).
3. Clear USBCMD[RS] (run/stop) and wait for USBSTS[HCH] to transition to a one. In Device mode, the Test Mode starts only if Run/Stop bit is set to 1. In Host mode, the Test Mode starts regardless of Run/Stop bit.
4. Set the Port Test Control field in the port under test PORTSC register to the value corresponding to the desired test mode. If the selected test is Test_Force_Enable, then USBCMD[RS] must then be transitioned back to one, in order to enable transmission of SOFs out of the port under test.
5. When the test is complete, system software must ensure the host controller is halted (HCH bit is a one) then it terminates and exits test mode by setting USBCMD[RST].

16.6.14 Interrupts

The EHCI host controller hardware provides interrupt capability based on a number of sources. The following list describes the general groups of interrupt sources:

- Interrupts as a result of executing transactions from the schedule (success and error conditions),
- Host controller events (Port change events, etc.)
- Host controller error events

All transaction-based sources are maskable through the host controller's Interrupt Enable register (USBINTR). Additionally, individual transfer descriptors can be marked to generate an interrupt on completion. This section describes each interrupt source and the processing that occurs in response to the interrupt.

During normal operation, interrupts may be immediate or deferred until the next interrupt threshold occurs. The interrupt threshold is a tunable parameter via the interrupt threshold control field in the USBCMD register. The value of this register controls when the host controller generates an interrupt on behalf of normal transaction execution. When a transaction completes during an interrupt interval period, the interrupt signaling the completion of the transfer will not occur until the interrupt threshold occurs. For example, the default value is eight microframes. This means that the host controller will not generate interrupts any more frequently than once every eight microframes.

[Section 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 read the USBSTS. It then acknowledges the interrupt by clearing all of the interrupt status bits by writing ones to these bit positions. The handler then determines whether the interrupt is due to schedule processing or some other event. After acknowledging the interrupt, the handler (via an OS-specific mechanism), schedules a deferred procedure call (DPC) which will execute later. The DPC routine processes the results of the schedule execution. The precise mechanisms used are beyond the scope of this document.

NOTE

The only method software should use for acknowledging an interrupt is by transitioning the appropriate status bits in the USBSTS register from a one to a zero.

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 16-72 lists the events/responses that the host can observe as a result of a transaction. The effects of the error counter and interrupt status are summarized in the following paragraphs. Most of these errors set the XactErr status bit in the appropriate interface data structure.

Table 16-72. Summary of Transaction Errors

Event/ Result	Queue Head/qTD/iTD/siTD Side Effects		USBSTS[USBERRINT]
	Cerr	Status Field	
CRC	-1	XactErr set	1 ¹
Timeout	-1	XactErr set	1 ¹
Bad PID ²	-1	XactErr set	1 ¹
Babble	N/A	See Section 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.

USBSTS[UEI] (USB error interrupt) is set and if the USBINTR[UEE] (USB error interrupt enable) is set, then a hardware interrupt is signaled to the system at the next interrupt threshold. The host controller will never start an OUT transaction that babbles across a microframe EOF.

NOTE

When a host controller detects a data PID mismatch, it must either: disable the packet babble checking for the duration of the bus transaction or do packet babble checking based solely on Maximum Packet Size. The USB core specification defines the requirements on a data receiver when it receives a data PID mismatch (for example, expects a DATA0 and gets a DATA1 or visa-versa). In summary, it must ignore the received data and respond with an ACK handshake, in order to advance the transmitter's data sequence. The EHCI interface allows system software to provide buffers for a Control, Bulk or Interrupt IN endpoint that are not an even multiple of the maximum packet size specified by the device. Whenever a device misses an ACK for an IN endpoint, the host and device are out of synchronization with respect to the progress of the data transfer. The host controller may have advanced the transfer to a buffer that is less than maximum packet size. The device re-sends its maximum packet size data packet, with the original data PID, in response to the next IN token. In order to properly manage the bus protocol, the host controller must disable the packet babble check when it observes the data PID mismatch.

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 USBSTS[UI] (USB interrupt) to be set. In addition, if a short packet is encountered on an IN transaction associated with a queue head, then this event also causes USBINT to be set. If USBINTR[UE] (USB interrupt enable) is set, a hardware interrupt is signaled to the system at the next interrupt threshold. If the completion is because of errors, USBSTS[UEI] (USB error interrupt) is also set.

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, USBSTS[UI] (USB interrupt bit) is set. If the USB interrupt enable bit is set (USBINTR[UE]), a hardware interrupt is signaled to the system at the next interrupt threshold.

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. If the port change interrupt enable bit (PCE) in the USBINTR register is set, the host controller issues a hardware interrupt. The port status change bits in PORTSC include:

- Connect change status (CSC)
- Port enable/disable change (PEC)
- Over-current change (OCC)
- Force port resume (FPR)

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, USBSTS[FRI]. If USBINTR[FRE] is set (frame list rollover enable), the host controller issues a hardware interrupt. This interrupt is not delayed to the next interrupt threshold.

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 USBCMD[IAA]. If it is set, it sets USBSTS[AAI]. If USBINTR[AAE] is set, the host controller issues a hardware interrupt at the next interrupt threshold. A detailed explanation of this feature is described in [Section 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:

- USBCMD[RS] is cleared.

- USBSTS[SEI] and USBSTS[HCH] register are set
- If the host system error enable bit, USBINTR[SEE] is set, the host controller issues a hardware interrupt. This interrupt is not delayed to the next interrupt threshold.

Table 16-73 summarizes the required actions taken on the various host errors.

Table 16-73. Summary Behavior on Host System Errors

Cycle Type	Master Abort	Target Abort	Data Phase Parity
Frame list pointer fetch (read)	Fatal	Fatal	Fatal
siTD fetch (read)	Fatal	Fatal	Fatal
siTD status write-back (write)	Fatal	Fatal	Fatal
iTD fetch (read)	Fatal	Fatal	Fatal
iTD status write-back (write)	Fatal	Fatal	Fatal
qTD fetch (read)	Fatal	Fatal	Fatal
qHD status write-back (write)	Fatal	Fatal	Fatal
Data write	Fatal	Fatal	Fatal
Data read	Fatal	Fatal	Fatal

NOTE

After a host system error, software must reset the host controller using USBCMD[RST] before re-initializing and restarting the host controller.

16.7 Device Data Structures

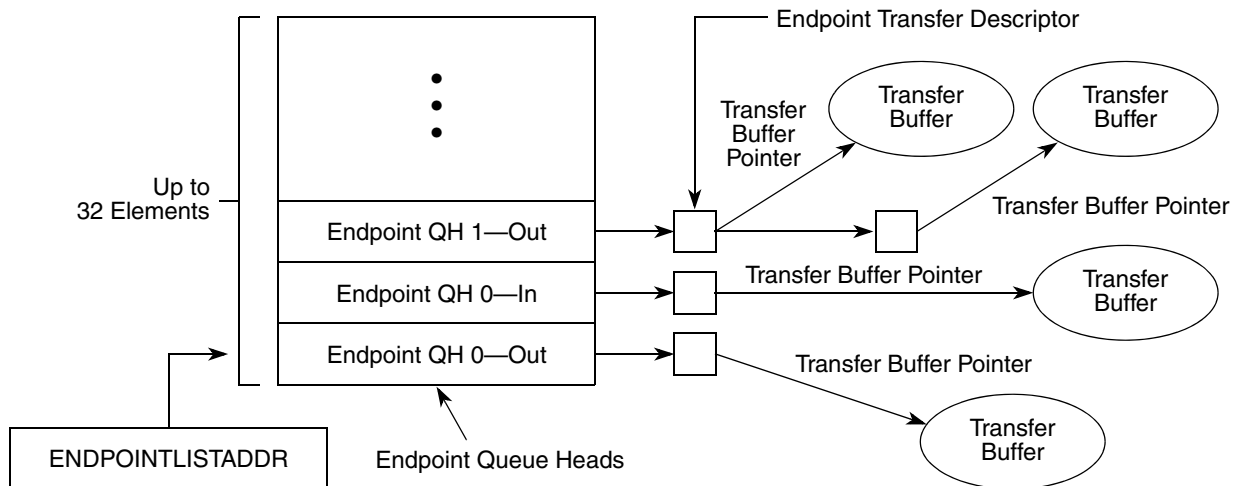
This section defines the interface data structures used to communicate control, status, and data between device controller driver (DCD) software and the device controller. The data structure definitions in this chapter support a 32-bit memory buffer address space. The interface consists of device queue heads and transfer descriptors.

NOTE

Software must ensure that no interface data structure reachable by the device controller spans a 4K-page boundary.

The data structures defined in the section are (from the device controller's perspective) a mix of read-only and read/writable fields. The device controller must preserve the read-only fields on all data structure writes.

The USB DR module includes DCD software called the USB 2.0 Device API. The device API provides an easy to use Application Program Interface for developing device (peripheral) applications. The device API incorporates and abstracts for the application developer all of the elements of the program interface.


Figure 16-60. Endpoint 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 16-61 shows the Endpoint Queue Head structure.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	offset
Mult	zlt	00	Maximum Packet Length													ios	000_0000_0000_0000										0x00					
Current dTD Pointer ¹																0_0000												0x04				
Next dTD Pointer ¹																0000										T ¹	0x08 ²					
0	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
Setup Buffer Bytes 3–0 ¹																																0x28
Setup Buffer Bytes 7–4 ¹																																0x2C

Figure 16-61. 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 16-74 describes the endpoint capabilities and characteristics fields.

Table 16-74. Endpoint Capabilities/Characteristics

Bits	Name	Description
31–30	Mult	Mult. This field is used to indicate the number of packets executed per transaction description as given by the following: 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 the total transfer length is a multiple. This bit is not relevant for Isochronous transfers. 0 Enable zero length packet to terminate transfers equal to a multiple of the Maximum Packet Length. (default). 1 Disable the zero length packet on transfers that are equal in length to a multiple Maximum Packet Length.
28–27	—	Reserved, 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.

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 is copied into this queue head overlay area by the device controller. Until a transfer is expired, software must not write the queue head overlay area or the associated transfer descriptor. When the transfer is complete, the device controller will write the results back to the original transfer descriptor and advance the queue.

See dTD for a description of the overlay fields.

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 controller (hardware) use only and should not be modified by DCD software.

Table 16-75 describes the current dTD pointer fields.

Table 16-75. Current dTD Pointer

Bits	Description
31–5	Current dtd. This field is a pointer to the dTD that is represented in the transfer overlay area. This field is modified by the Device Controller to next dTD pointer during endpoint priming or queue advance.
4–0	Reserved, should be cleared. Bit reserved for future use and should be cleared.

16.7.1.4 Setup Buffer

The setup buffer is dedicated storage for the 8-byte data that follows a setup PID.

NOTE

Each endpoint has a TX and an RX dQH associated with it, and only the RX queue head is used for receiving setup data packets.

Table 16-76 describes the multiple mode control fields.

Table 16-76. Multiple Mode Control

DWord	Bits	Description
1	31–0	Setup Buffer 0. This buffer contains bytes 3 to 0 of an incoming setup buffer packet and is written by the device controller to be read by software.
2	31–0	Setup Buffer 1. This buffer contains bytes 7 to 4 of an incoming setup buffer packet and is written by the device controller to be read by software.

16.7.2 Endpoint Transfer Descriptor (dTD)

The dTD describes the location and quantity of data to be sent/received for given transfer to the device controller. The DCD should not attempt to modify any field in an active dTD except the Next Link Pointer, which should only be modified as described in [Section 16.8.5, “Managing Transfers with Transfer Descriptors.”](#)

Figure 16-62 shows the endpoint transfer descriptor.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	offset
Next Link Pointer																												0000	T	0x00		
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 16-62. Endpoint Transfer Descriptor (dTD)

¹ Device controller read/write; all others read-only.

Table 16-77 describes the next dTD pointer fields.

Table 16-77. Next dTD Pointer

Bits	Description
31–5	Next transfer element pointer. This field contains the physical memory address of the next dTD to be processed. The field corresponds to memory address signals [31:5], respectively.
4–1	Reserved, should be cleared. Bits reserved for future use and should be cleared.
0	Terminate (T). 1=pointer is invalid. 0=Pointer is valid (points to a valid Transfer Element Descriptor). This bit indicates to the Device Controller that there are no more valid entries in the queue.

Table 16-78 describes the next dTD token fields.

Table 16-78. dTD Token

Bits	Description
31	Reserved, should be cleared. Bit reserved for future use and should be cleared.
30–16	Total Bytes. This field specifies the total number of bytes to be moved with this transfer descriptor. This field is decremented by the number of bytes actually moved during the transaction and only on the successful completion of the transaction. 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 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 than 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.

Table 16-78. dTD Token (continued)

Bits	Description
11–10	Multiplier Override (MultiO). This field can be used for transmit ISO's (that is, ISO-IN) to override the multiplier in the QH. This field must be zero for all packet types that are not transmit-ISO. 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 MultiO = greatest integer of (Total Bytes/Max. Packet Size) except for the case when Total bytes = 0; then MultiO should be 1. Note: Non-ISO and non-TX endpoints must set MultiO = 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, should be cleared

Table 16-79–Table 16-81 describes the buffer pointer page *n* fields.

Table 16-79. Buffer Pointer Page 0

Bits	Description
31–12	Buffer Pointer. Selects the page offset in memory for the packet buffer. Non virtual memory systems will typically set the buffer pointers to a series of incrementing integers.
11–0	Current Offset. Offset into the 4kb buffer where the packet is to begin.

Table 16-80. Buffer Pointer Page 1

Bits	Description
31–12	Buffer Pointer. Selects the page offset in memory for the packet buffer. Non virtual memory systems will typically set the buffer pointers to a series of incrementing integers.
11	Reserved
10–0	Frame Number. Written by the device controller to indicate the frame number in which a packet finishes. This is typically be used to correlate relative completion times of packets on an ISO endpoint.

Table 16-81. Buffer Pointer Pages 2–4

Bits	Description
31–12	Buffer Pointer. Selects the page offset in memory for the packet buffer. Non virtual memory systems will typically set the buffer pointers to a series of incrementing integers.
11–0	Reserved

16.8 Device Operational Model

The function of the device operation is to transfer a request in the memory image to and from the Universal Serial Bus. Using a set of linked list transfer descriptors, pointed to by a queue head, the device controller will perform the data transfers. The following sections explain the use of the device controller from the device controller driver (DCD) point-of-view and further describe how specific USB bus events relate to status changes in the device controller programmer's interface.

16.8.1 Device Controller Initialization

After hardware reset, the USB DR module is disabled until the run/stop bit (USBCMD[RS]) is set to a '1'. In the disabled state, the pull-up on the USB D+ is not active which prevents an attach event from occurring. At a minimum, it is necessary to have the queue heads setup for endpoint zero before the device attach occurs. Shortly after the device is enabled, a USB reset will occur followed by setup packet arriving at endpoint 0. A queue head must be prepared so that the device controller can store the incoming setup packet.

To configure the external ULPI PHY the following initialization sequence is required.

1. After power-on reset the UTMI PHY will be in disabled state and the PLL will be held reset. The UTMI PHY should remain disabled if the ULPI is being used.
2. Set the CONTROL[PHY_CLK_SEL] bits to select the ULPI PHY as the source of USB controller PHY clock.
3. Wait for PHY clock to become valid. This can be determined by polling the CONTROL[PHY_CLK_VALID] status bit. Note that this bit is not valid once the CONTROL[USB_EN] bit is set.

Once the PHY clock is valid the user can proceed to the device controller initialization phase.

In order to initialize a device, the software should perform the following steps:

1. Set the controller mode to device mode. Optionally set USBMODE[SDIS] (streaming disable).

NOTE

Transitioning from host mode to device mode requires a device controller reset before modifying USBMODE.

2. Optionally modify the BURSTSIZE register.
3. Program PORTSC[PTS] if using a non-ULPI PHY.
4. Set CONTROL[USB_EN]
5. Allocate and initialize device queue heads in system memory Minimum: Initialize device queue heads 0 Tx and 0 Rx.

NOTE

All device queue heads must be initialized for control endpoints before the endpoint is enabled. Device queue heads for non-control endpoints must be initialized before the endpoint can be used.

For information on device queue heads, refer to [Section 16.7, “Device Data Structures.”](#)

6. Configure the ENDPOINTLISTADDR pointer.
For additional information on ENDPOINTLISTADDR, refer to the register table.
7. Enable the microprocessor interrupt associated with the USB DR module and optionally change setting of USBCMD[ITC].
Recommended: enable all device interrupts including: USBINT, USBERRINT, Port Change Detect, USB Reset Received, DCSuspend.
For a list of available interrupts refer to the USBINTR and the USBSTS register tables.
8. Set USBCMD[RS] to run mode.
After the run bit is set, a device reset will occur. The DCD must monitor the reset event and set the DEVICEADDR register, set the ENDPTCTRLx registers, and adjust the software state as described in [Section 16.8.2.1, “Bus Reset.”](#)

NOTE

Endpoint 0 is designed as a control endpoint only and does not need to be configured using ENDPTCTRL0 register.

It is also not necessary to initially prime Endpoint 0 because the first packet received will always be a setup packet. The contents of the first setup packet will require a response in accordance with USB device framework command set.

16.8.2 Port State and Control

From a chip or system reset, the USB_DR controller enters the powered state. A transition from the powered state to the attach state occurs when the run/stop bit (USBCMD[RS]) is set to a ‘1’. After receiving a reset on the bus, the port will enter the defaultFS or defaultHS state in accordance with the protocol reset described in Appendix C.2 of the *USB Specification Rev. 2.0*. [Figure 16-63](#) depicts the state of a USB 2.0 device.

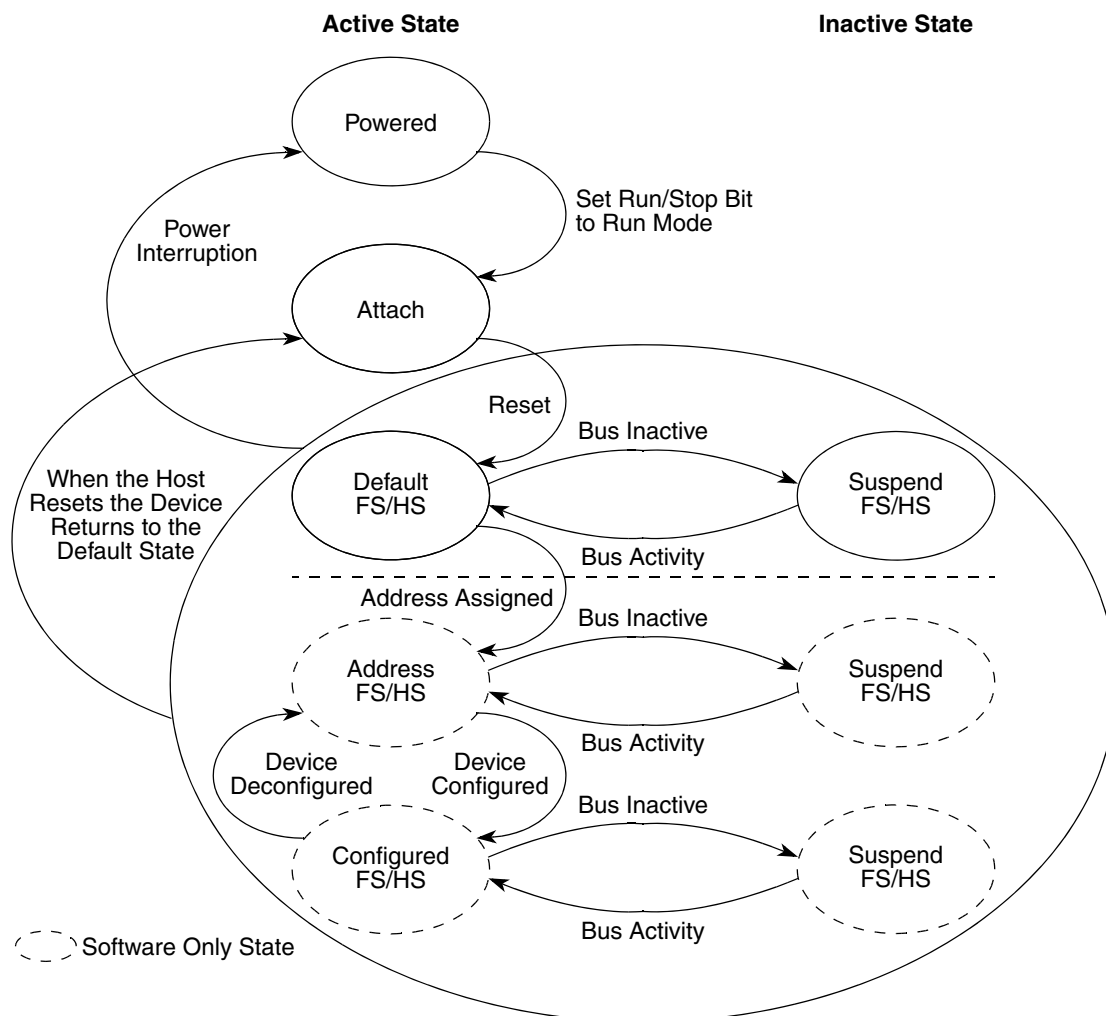


Figure 16-63. USB 2.0 Device States

States Powered, Attach, DefaultFS/HS, SuspendFS/HS are implemented in the USB_DR controller and are communicated to the DCD using status bits, as shown in [Table 16-82](#):

Table 16-82. Device Controller State Information Bits

Bits	Register
DCSuspend (SLI)	USBSTS
USB Reset Received (URI)	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 `ENDPTCTRLn` 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 bit, `USBINTR[URE]`, is set). After a reset is received, all endpoints (except endpoint 0) are disabled and any primed transactions are canceled by the device controller. The concept of priming is clarified below, but the DCD must perform the following tasks when a reset is received:

1. Clear all setup token semaphores by reading the `ENDPTSETUPSTAT` register and writing the same value back to the `ENDPTSETUPSTAT` register.
2. Clear all the endpoint complete status bits by reading the `ENDPTCOMPLETE` register and writing the same value back to the `ENDPTCOMPLETE` register.
3. Cancel all primed status by waiting until all bits in the `ENDPTPRIME` are 0 and then writing `0xFFFF_FFFF` to `ENDPTFLUSH`.
4. Read the reset bit in the `PORTSC` register (`PORTSC[PR]`) and make sure that it is still active.
 - A USB reset occurs for a minimum of 3 ms, and the DCD must reach this point in the reset cleanup before end of the reset occurs.
 - If it does not, a hardware reset of the device controller is recommended. A hardware reset can be performed by writing a one to the `USB_DR` controller reset bit in (`USBCMD[RST]`). Note that a hardware reset will cause the device to detach from the bus by clearing `USBCMD[RS]` bit. Thus, the DCD must completely re-initialize the `USB_DR` controller after a hardware reset.
5. Free all allocated dTDs because they will no longer be executed by the device controller. If this is the first time the DCD is processing a USB reset event, then it is likely that no dTDs have been allocated.

At this time, the DCD may release control back to the OS because no further changes to the device controller are permitted until a Port Change Detect is indicated.

After a Port Change Detect, the device has reached the default state and the DCD can read the `PORTSC` to determine if the device is operating in FS or HS mode. At this time, the device controller has reached normal operating mode and DCD can begin enumeration according to the USB Chapter 9, Device Framework.

NOTE

The device DCD may use the FS/HS mode information to determine the bandwidth mode of the device.

In some applications, it may not be possible to enable one or more pipes while in FS mode. Beyond the data rate issue, there is no difference in DCD operation between FS and HS modes.

16.8.2.2 Suspend/Resume

This section discusses the suspend and resume functions.

16.8.2.2.1 Suspend Description

In order to conserve power, USB_DR controller automatically enters the suspended state when no bus traffic has been observed for a specified period. When suspended, the USB_DR controller maintains any internal status, including its address and configuration. Attached devices must be prepared to suspend at any time they are powered, regardless of if they have been assigned a non-default address, are configured, or neither. Bus activity may cease due to the host entering a suspend mode of its own. In addition, a USB device shall also enter the suspended state when the hub port it is attached to is disabled.

The USB_DR controller exits suspend mode when there is bus activity. It may also request the host to exit suspend mode or selective suspend using electrical signaling to indicate remote wake-up. The ability of a device to signal remote wake-up is optional. The USB_DR controller is capable of remote wake-up signaling. When the USB_DR controller is reset, remote wake-up signaling must be disabled.

16.8.2.2.2 Suspend Operational Model

The USB_DR controller moves into the suspend state when suspend signaling is detected or activity is missing on the upstream port for more than a specific period. After the device controller enters the suspend state, the DCD is notified by an interrupt (assuming DC Suspend Interrupt is enabled). When the USBSTS[SLI] (device controller suspend) is set, the device controller is suspended.

DCD response when the device controller is suspended is application specific and may involve switching to low power operation. Information on the bus power limits in suspend state can be found in USB 2.0 specification.

16.8.2.2.3 Resume

If the USB_DR controller is suspended, its operation is resumed when any non-idle signaling is received on its upstream facing port. In addition, the USB_DR controller can signal the system to resume operation by forcing resume signaling to the upstream port. Resume signaling is sent upstream by writing a '1' to the PORTSC[FPR] (resume bit) while the device is in suspend state. Sending resume signal to an upstream port should cause the host to issue resume signaling and bring the suspended bus segment (one or more devices) back to the active condition.

NOTE

Before resume signaling can be used, the host must enable it using the Set Feature command defined in device framework (Chapter 9) of the USB 2.0 Specification.

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 controller supports up to three endpoint specified numbers. The DCD can enable, disable, and configure each endpoint.

Each endpoint direction is essentially independent and can be configured with differing behavior in each direction. For example, the DCD can configure endpoint 1-IN to be a bulk endpoint and endpoint 1-OUT to be an isochronous endpoint. This helps to conserve the total number of endpoints required for device operation. The only exception is that control endpoints must use both directions on a single endpoint number to function as a control endpoint. Endpoint 0 is, for example, is always a control endpoint and uses the pair of directions.

Each endpoint direction requires a queue head allocated in memory. If the maximum of 6 endpoint numbers, one for each endpoint direction are being used by the device controller, then 12 queue heads are required. The operation of an endpoint and use of queue heads are described later in this document.

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 `ENDPTCTRLn` register. Each 32-bit `ENDPTCTRLn` is split into an upper and lower half. The lower half of `ENDPTCTRLn` 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 `ENDPTCTRLn` register otherwise the behavior is undefined. [Table 16-83](#) shows how to construct a configuration word for endpoint initialization.

Table 16-83. 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 controller may need to return to the host a STALL.

The first occasion is the functional stall, which is a condition set by the DCD as described in the USB 2.0 device framework (Chapter 9). A functional stall is only used on non-control endpoints and can be enabled in the device controller by setting the endpoint stall bit in the `ENDPTCTRLn` register associated with the given endpoint and the given direction. In a functional stall condition, the device controller will continue to return STALL responses to all transactions occurring on the respective endpoint and direction until the endpoint stall bit is cleared by the DCD.

A protocol stall, unlike a function stall, is used on control endpoints is automatically cleared by the device controller at the start of a new control transaction (setup phase). When enabling a protocol stall, the DCD should enable the stall bits (both directions) as a pair. A single write to the `ENDPTCTRLn` register can ensure that both stall bits are set at the same instant.

NOTE

Any write to the `ENDPTCTRLn` register during operational mode must preserve the endpoint type field (that is, perform a read-modify-write).

Table 16-84 describes the device controller stall response matrix.

Table 16-84. Device Controller Stall Response Matrix

USB Packet	Endpoint Stall Bit.	Effect on STALL Bit.	USB Response
SETUP packet received by a non-control endpoint	N/A	None	STALL
IN/OUT/PING packet received by a non-control endpoint	'1'	None	STALL
IN/OUT/PING packet received by a non-control endpoint	'0'	None	ACK/NAK/NYET
SETUP packet received by a control endpoint	N/A	Cleared	ACK
IN/OUT/PING packet received by a control endpoint	'1'	None	STALL
IN/OUT/PING packet received by a control endpoint	'0'	None	ACK/NAK/NYET

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 *Universal Serial Bus Revision 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 `ENDPTCTRLn` 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 controller to ignore the data toggle pattern that is normally sent and accept all incoming data packets regardless of the data toggle state.

In normal operation, the USB_DR controller checks the DATA0/DATA1 bit against the data toggle state bit 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 controller from re-sending the same packet, the device controller will respond to the error packet by acknowledging it with either an ACK or NYET response.

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 *Universal Serial Bus Revision 2.0 Specification*.

A USB host will send requests to the USB_DR controller in an order that cannot be precisely predicted as a single pipeline, so it is not possible to prepare a single packet for the device controller to execute. However, the order of packet requests is predictable when the endpoint number and direction is considered. For example, if endpoint 2 (transmit direction) is configured as a bulk pipe, then expect the host will send IN requests to that endpoint. This USB_DR controller prepares packets for each endpoint/direction in anticipation of the host request. The process of preparing the device controller to send or receive data in response to host initiated transaction on the bus is referred to as ‘priming’ the endpoint. This term is used throughout the following documentation to describe the USB_DR controller operation so the DCD can be architected properly use priming. Further, note that the term ‘flushing’ is used to describe the action of clearing a packet that was queued for execution.

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 is stored in the dQH until the device controller completes the transfer described by the dTd. Storing the dTd in the dQH allows the device controller to fetch the operating context needed to handle a request from the host without the need to follow the linked list, starting at the dQH when the host request is received.

After the device has loaded the dTd, the leading data in the packet is stored in a FIFO in the device controller. This FIFO is split into virtual channels so that the leading data can be stored for any endpoint .

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 has been 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 is retired by the device controller when the packets described in the transfer descriptor have been completed. Each dTD describes N packets to be transferred according to the USB Variable Length transfer protocol. The formula, [Table 16-85](#), and [Table 16-86](#) describe how the device controller computes the number and length of the packets to be sent/received by the USB vary according to the total number of bytes and maximum packet length.

With Zero Length Termination (ZLT) = 0

$$N = \text{INT}(\text{number of bytes}/\text{max. packet length}) + 1$$

With Zero Length Termination (ZLT) = 1

$$N = \text{MAXINT}(\text{number of bytes}/\text{max. packet length})$$

Table 16-85. Variable Length Transfer Protocol Example (ZLT = 0)

Bytes (dTD)	Max. Packet Length (dQH)	N	P1	P2	P3
511	256	2	256	255	—
512	256	3	256	256	0
512	512	2	512	0	—

Table 16-86. Variable Length Transfer Protocol Example (ZLT = 1)

Bytes (dTD)	Max. Packet Length (dQH)	N	P1	P2	P3
511	256	2	256	255	—
512	256	2	256	256	—
512	512	1	512	—	—

NOTE

The MULT field in the dQH must be set to ‘00’ for bulk, interrupt, and control endpoints.

TX-dTD is complete when:

- All packets described dTD were successfully transmitted. *** Total bytes in dTD will equal zero when this occurs.

RX-dTD is complete when:

- All packets described in dTD were successfully received. *** Total bytes in dTD will equal zero when this occurs.
- A short packet (number of bytes < maximum packet length) was received. *** This is a successful transfer completion; DCD must check Total Bytes in dTD to determine the number of bytes that are remaining. From the total bytes remaining in the dTD, the DCD can compute the actual bytes received.
- A long packet was received (number of bytes > maximum packet size) OR (total bytes received > total bytes specified). *** This is an error condition. The device controller will discard the remaining packet, and set the Buffer Error bit in the dTD. In addition, the endpoint is flushed and the USBERR interrupt will become active.

On the successful completion of the packet(s) described by the dTD, the active bit in the dTD is cleared and the next pointer will be followed when the Terminate bit is clear. When the Terminate bit is set, the USB_DR controller will flush the endpoint/direction and cease operations for that endpoint/direction.

On the unsuccessful completion of a packet (see long packet above), the dQH is left pointing to the dTD that was in error. In order to recover from this error condition, the DCD must properly re-initialize the dQH by clearing the active bit and update the nextTD pointer before attempting to re-prime the endpoint.

NOTE

All packet level errors such as a missing handshake or CRC error will be retried automatically by the device controller.

There is no required interaction with the DCD for handling such errors.

16.8.3.4.1 Interrupt/Bulk Endpoint Bus Response Matrix

Table 16-87 shows the interrupt/bulk endpoint bus response matrix.

Table 16-87. Interrupt/Bulk Endpoint Bus Response Matrix

	Stall	Not Primed	Primed	Underflow	Overflow
Setup	Ignore	Ignore	Ignore	N/A	N/A
In	STALL	NAK	Transmit	BS Error ¹	N/A
Out	STALL	NAK	Receive + NYET/ACK ²	N/A	NAK
Ping	STALL	NAK	ACK	N/A	N/A
Invalid	Ignore	Ignore	Ignore	Ignore	Ignore

¹ Force Bit Stuff Error.

² NYET/ACK—NYET unless the Transfer Descriptor has packets remaining according to the USB variable length protocol then ACK.

SYSERR—System error should never occur when the latency FIFOs are correctly sized and the DCD is responsive.

16.8.3.5 Control Endpoint Operation Model

This section discusses the 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 controller will always accept the setup phase unless the setup lockout is engaged.

The setup lockout will engage so that future setup packets are ignored. Lockout of setup packets ensures that while software is reading the setup packet stored in the queue head, that data is not written as it is being read potentially causing an invalid setup packet.

The setup lockout mechanism can be disabled and a tripwire type semaphore will ensure that the setup packet payload is extracted from the queue head without being corrupted by an incoming setup packet. This is the preferred behavior because ignoring repeated setup packets due to long software interrupt latency would be a compliance issue.

Setup Packet Handling

- Disable Setup Lockout by writing '1' to Setup Lockout Mode (SLOM) in USBMODE (once at initialization). Setup lockout is not necessary when using the tripwire as described below.

NOTE

Leaving the Setup Lockout Mode as '0' will result in a potential compliance issue.

- After receiving an interrupt and inspecting ENDPTSETUPSTAT to determine that a setup packet was received on a particular pipe:
 - Write '1' to clear corresponding bit ENDPTSETUPSTAT.
 - Write '1' to Setup Tripwire (SUTW) in USBCMD register.
 - Duplicate contents of dQH.SetupBuffer into local software byte array.
 - Read Setup TripWire (SUTW) in USBCMD register. (if set—continue; if cleared—goto 2)
 - Write '0' to clear Setup Tripwire (SUTW) in USBCMD register.
 - Process setup packet using local software byte array copy and execute status/handshake phases.

NOTE

After receiving a new setup packet the status and/or handshake phases may still be pending from a previous control sequence. These should be flushed and de-allocated before linking a new status and/or handshake dTD for the most recent setup packet.

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

Table 16-88 shows the device controller response to packets on a control endpoint, according to the device controller state.

Table 16-88. Control Endpoint Bus Response Matrix

Token Type	Endpoint State					Setup Lockout
	Stall	Not Primed	Primed	Underflow	Overflow	
Setup	ACK	ACK	ACK	N/A	SYSEERR ¹	
In	STALL	NAK	Transmit	BS Error ²	N/A	N/A
Out	STALL	NAK	Receive + NYET/ACK ³	N/A	NAK	N/A

Table 16-88. Control Endpoint Bus Response Matrix (continued)

Token Type	Endpoint State					Setup Lockout
	Stall	Not Primed	Primed	Underflow	Overflow	
Ping	STALL	NAK	ACK	N/A	N/A	N/A
Invalid	Ignore	Ignore	Ignore	Ignore	Ignore	Ignore

¹ SYSERR—System error should never occur when the latency FIFOs are correctly sized and the DCD is responsive.

² Force Bit Stuff Error.

³ NYET/ACK—NYET unless the Transfer Descriptor has packets remaining according to the USB variable length protocol then ACK.

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 controller is accomplished by the following:

- Exactly MULT Packets per (micro)Frame are transmitted/received. Note that MULT is a two-bit field in the device Queue Head. The variable length packet protocol is not used on isochronous endpoints.
- NAK responses are not used. Instead, zero length packets 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 is held ready until executed or canceled by the DCD.

The USB_DR controller in host mode uses the periodic frame list to schedule data exchanges to Isochronous endpoints. The operational model for device mode does not use such a data structure. Instead, the same dTD used for Control/Bulk/Interrupt endpoints is also used for isochronous endpoints. The difference is in the handling of the dTD.

The first difference between bulk and ISO-endpoints is that priming an ISO-endpoint is a delayed operation such that an endpoint will become primed only after a SOF is received. After the DCD writes the prime bit, the prime bit is 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 controller will prime the isochronous endpoint in (micro)frame N – 1 so that the device controller will execute delivery during (micro)frame N.

CAUTION

Priming an endpoint towards the end of (micro)frame N – 1 will not guarantee delivery in (micro)frame N. The delivery may actually occur in (micro)frame N + 1 if device controller does not have enough time to complete the prime before the SOF for packet N is received.

16.8.3.6.2 Isochronous Endpoint Bus Response Matrix

Table 16-89 shows the isochronous endpoint bus response matrix.

Table 16-89. Isochronous Endpoint Bus Response Matrix

	Stall	Not Primed	Primed	Underflow	Overflow
Setup	STALL	STALL	STALL	N/A	N/A
In	NULL ¹ Packet	NULL Packet	Transmit	BS Error ²	N/A
Out	Ignore	Ignore	Receive	N/A	Drop Packet
Ping	Ignore	Ignore	Ignore	Ignore	Ignore
Invalid	Ignore	Ignore	Ignore	Ignore	Ignore

¹ Zero Length Packet.

² Force Bit Stuff Error.

16.8.4 Managing Queue Heads

Figure 16-64 shows the endpoint queue head diagram.

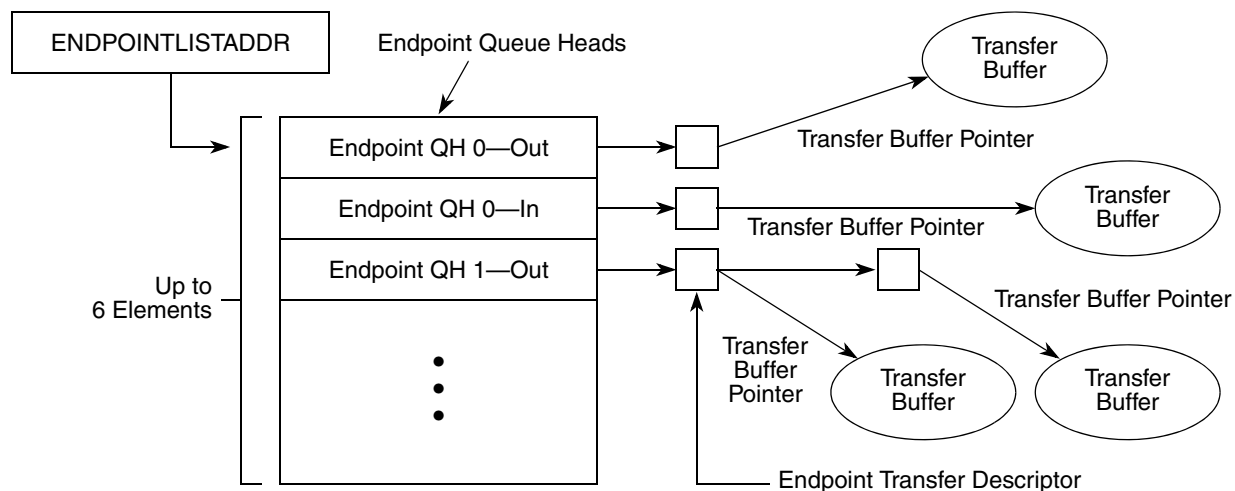


Figure 16-64. Endpoint Queue Head Diagram

The device queue head (dQH) points to the linked list of transfer tasks, each depicted by the device Transfer Descriptor (dTDD). An area of memory pointed to by ENDPOINTLISTADDR contains a group of all dQHs in a sequential list as shown in [Figure 16-64](#). The even elements in the list of dQH's are used for receive endpoints (OUT/SETUP) and the odd elements are used for transmit endpoints (IN/INTERRUPT). Device transfer descriptors are linked head to tail starting at the queue head and ending at a terminate bit. Once the dTD has been retired, it will no longer be part of the linked list from the queue head. Therefore, software is required to track all transfer descriptors since pointers will no longer exist within the queue head once the dTD is retired (see [Section 16.8.5.1, "Software Link Pointers"](#)).

In addition to the current and next pointers and the dTD overlay, the dQH also contains the following parameters for the associated endpoint: Multiplier, Maximum Packet Length, Interrupt On Setup. The complete initialization of the dQH including these fields is demonstrated in the next section.

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 MaxPacketSize field as required by the USB Chapter 9 or application specific protocol.
- Write the multiplier field to 0 for control, bulk, and interrupt endpoints. For ISO endpoints, set the multiplier to 1, 2, or 3 as required bandwidth and in conjunction with the USB Chapter 9 protocol. Note that in FS mode, the multiplier field can only be 1 for ISO endpoints.
- Write the next dTD Terminate bit field to '1.'
- Write the Active bit in the status field to '0.'
- Write the Halt bit in the status field to '0.'

NOTE

The DCD must only modify dQH if the associated endpoint is not primed and there are no outstanding dTDs.

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 16.8.5.5, "Flushing/De-Priming an Endpoint."](#)

NOTE

It is possible for the device controller to receive setup packets before previous control transfers complete. Existing control packets in progress must be flushed and the new control packet completed.

4. Decode setup packet and prepare data phase [optional] and status phase transfer as required by the USB Chapter 9 or application specific protocol.

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 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 16-65 shows the software link pointers.

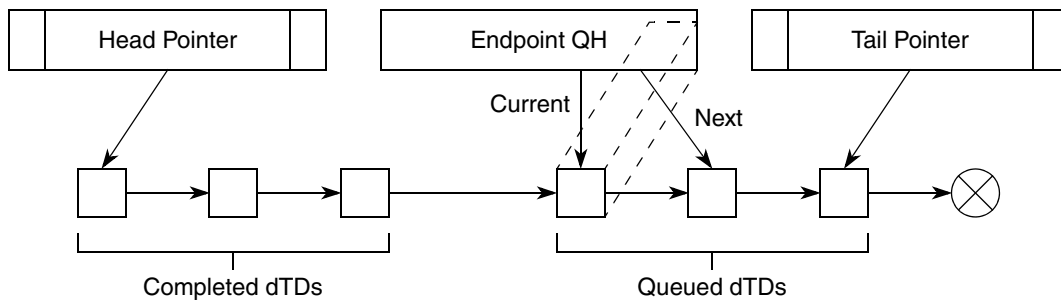


Figure 16-65. Software Link Pointers

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 seven DWords to ‘0’.
2. Set the terminate bit to ‘1’.
3. Fill in total bytes with transfer size.
4. Set the interrupt on complete if desired.
5. Initialize the status field with the active bit set to ‘1’ and all remaining status bits set to ‘0’.
6. Fill in buffer pointer page 0 and the current offset to point to the start of the data buffer.
7. Initialize buffer pointer page 1 through page 4 to be one greater than each of the previous buffer pointer.

16.8.5.3 Executing a Transfer Descriptor

To safely add a dTD, the DCD must account for the event in which the device controller reaches the end of the dTD list at the same time a new dTD is being added to the end of the list.

First, determine whether the link list is empty by checking the DCD driver to see if the pipe is empty (internal representation of linked-list should indicate if any packets are outstanding). Then follow the sequence of actions in the following list as appropriate, depending on whether the link list is empty or not empty.

- 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 (4) is ‘1’ DONE.
 8. If status bit read in (4) 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 is notified with a USB interrupt if the Interrupt On

Complete bit was set or alternately, the DCD can poll the endpoint complete register to find when the dTD had been executed. After a dTD has been executed, DCD can check the status bits to determine success or failure.

CAUTION

Multiple dTD can be completed in a single endpoint complete notification. After clearing the notification, DCD must search the dTD linked list and retire all dTDs that have finished (Active bit cleared).

By reading the status fields of the completed dTDs, the DCD can determine if the transfers completed successfully. Success is determined with the following combination of status bits:

- Active = 0
- Halted = 0
- Transaction Error = 0
- Data Buffer Error = 0

Should any combination other than the one shown above exist, the DCD must take proper action. Transfer failure mechanisms are indicated in the Device Error Matrix.

In addition to checking the status bit the DCD must read the Transfer Bytes field to determine the actual bytes transferred. When a transfer is complete, the Total Bytes transferred is by decremented by the actual bytes transferred. For Transmit packets, a packet is only complete after the actual bytes reaches zero, but for receive packets, the host may send fewer bytes in the transfer according the USB variable length packet protocol.

16.8.5.5 Flushing/De-Priming an Endpoint

It is necessary for the DCD to flush to de-prime one or more endpoints on a USB device reset or during a broken control transfer. There may also be application specific requirements to stop transfers in progress. The following procedure can be used by the DCD to stop a transfer in progress:

1. Write a '1' to the corresponding bit(s) in ENDPTFLUSH.
2. Wait until all bits in ENDPTFLUSH are '0'.
3. Software note: this operation may take a large amount of time depending on the USB bus activity. It is not desirable to have this wait loop within an interrupt service routine.
4. Read ENDPTSTATUS to ensure that for all endpoints commanded to be flushed, that the corresponding bits are now '0'. If the corresponding bits are '1' after step #2 has finished, then the flush failed as described in the following:

Explanation: In very rare cases, a packet is in progress to the particular endpoint when commanded flush using ENDPTFLUSH. A safeguard is in place to refuse the flush to ensure that the packet in progress completes successfully. The DCD may need to repeatedly flush any endpoints that fail to flush by repeating steps 1–3 until each endpoint is successfully flushed.

16.8.5.6 Device Error Matrix

Table 16-90 summarizes packet errors that are not automatically handled by the USB controller.

Table 16-90. Device Error Matrix

Error	Direction	Packet Type	Data Buffer Error Bit	Transaction Error Bit
Overflow **	RX	Any	1	0
ISO Packet Error	RX	ISO	0	1
ISO Fulfillment Error	Both	ISO	0	1

Notice that the device controller handles all errors on Bulk/Control/Interrupt Endpoints except for a data buffer overflow. However, for ISO endpoints, errors packets are not retried and errors are tagged as indicated. Table 16-91 provides the error descriptions.

Table 16-91. Error Descriptions

Overflow	Number of bytes received exceeded max. packet size or total buffer length. Note: This error also sets the Halt bit in the dQH. If there are dTDs remaining in the linked list for the endpoint, they will not be executed.
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 shown in Table 16-92. The most important of these is listed first because the DCD must acknowledge a setup buffer in the timeliest manner possible.

Table 16-92. 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 16.8.4, “Managing Queue Heads”). Process setup packet according to USB 2.0 Chapter 9 or application specific protocol.

Table 16-92. Interrupt Handling Order (continued)

Execution Order	Interrupt	Action
1b	USB Interrupt ENDPTCOMPLETE	Handle completion of dTD as indicated in Section 16.8.4, “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 interrupts shown in [Table 16-93](#). These interrupts can be handled in any order because they do not occur often in comparison to the high-frequency interrupts.

Table 16-93. Low Frequency Interrupt Events

Interrupt	Action
Port Change	Change software state information.
Sleep Enable (Suspend)	Change software state information. Low power handling as necessary.
Reset Received	Change software state information. Abort pending transfers.

16.8.6.3 Error Interrupts

Error interrupts are the least frequently occurring events. They should be placed last in the interrupt service routine. [Table 16-94](#) shows the error interrupt events.

Table 16-94. Error Interrupt Events

Interrupt	Action
USB Error Interrupt	This error is redundant because it combines USB Interrupt and an error status in the dTD. The DCD will more aptly handle packet-level errors by checking dTD status field upon receipt of USB Interrupt (w/ ENDPTCOMPLETE).
System Error	Unrecoverable error. Immediate Reset of core; free transfers buffers in progress and restart the DCD.

16.9 Deviations from the EHCI Specifications

The host mode operation of the USB DR module is nearly EHCI-compatible with few minor differences. For the most part, the module conforms to the data structures and operations described in Section 3, “Data Structures,” and Section 4, “Operational Model,” in the EHCI specification. The particulars of the deviations occur in the following areas:

- Embedded transaction translator—Allows direct attachment of FS and LS devices in host mode without the need for a companion controller.

- Device operation—In host mode, the device operational registers are generally disabled and thus device mode is mostly transparent when in host mode. However, there are a couple exceptions documented in the following sections.
- 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

The DR module supports directly connected full and low speed devices without requiring a companion controller by including the capabilities of a USB 2.0 high speed hub transaction translator. Although there is no separate transaction translator block in the system, the transaction translator function normally associated with a high speed hub has been implemented within the DMA and Protocol engine blocks. The embedded transaction translator function is an extension to EHCI interface, but makes use of the standard data structures and operational models that exist in the EHCI specification to support full and low speed devices.

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 \(HSCPARAMS\),”](#) 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.
- Addition of two-bit Port Speed (PSPD) to the PORTSC 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 always sets the port enable after the port reset operation regardless of the result of the host device chirp result. The resulting port speed is indicated by the PSPD field in PORTSC. Therefore, the standard EHCI host controller driver requires an alteration to handle directly connected full- and

low-speed devices or hubs. [Table 16-95](#) summarizes the functional differences between EHCI and EHCI with embedded TT.

Table 16-95. Functional Differences Between EHCI and EHCI with Embedded TT

Standard EHCI	EHCI with Embedded Transaction Translator
After port enable bit is set following a connection and reset sequence, the device/hub is assumed to be HS.	After port enable bit is set following a connection and reset sequence, the device/hub speed is noted from PORTSC.
FS and LS devices are assumed to be downstream from a HS hub thus, all port-level control is performed through the Hub Class to the nearest Hub.	FS and LS device can be either downstream from a HS hub or directly attached. When the FS/LS device is downstream from a HS hub, then port-level control is done using the Hub Class through the nearest Hub. When a FS/LS device is directly attached, then port-level control is accomplished using PORTSC.
FS and LS devices are assumed to be downstream from a HS hub with HubAddr=X. [where HubAddr > 0 and HubAddr is the address of the Hub where the bus transitions from HS to FS/LS (that is, Split target hub)]	FS and LS device can be either downstream from a HS hub with HubAddr = X [HubAddr > 0] or directly attached [where HubAddr = 0 and HubAddr is the address of the Root Hub where the bus transitions from HS to FS/LS (that is, Split target hub is the root hub)]

16.9.1.4 Data Structures

The same data structures used for FS/LS transactions through a HS hub are also used for transactions through the Root Hub. The following list demonstrates how the Hub Address and Endpoint Speed fields should be set for directly attached FS/LS devices and hubs:

1. QH (for direct attach FS/LS)—Async. (Bulk/Control Endpoints) Periodic (Interrupt)
 - Hub Address = 0
 - Transactions to direct attached device/hub.
 - QH.EPS = Port Speed
 - Transactions to a device downstream from direct attached FS hub.
 - QH.EPS = Downstream Device Speed

NOTE

When QH.EPS = 01 (LS) and PORTSC[PSPD] = 00 (FS), a LS-pre-pid is 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 *Universal Serial Bus Revision 2.0 Specification*) and for the EHCI controller moving packets between system memory and a USB-HS hub. Since the embedded transaction translator exists within the DR module there is no physical

bus between EHCI host controller driver and the USB FS/LS bus. These sections will briefly discuss the operational model for how the EHCI and transaction translator operational models are combined without the physical bus between. The following sections assume the reader is familiar with both the EHCI and USB 2.0 transaction translator operational models.

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 *Universal Serial Bus Revision 2.0 Specification* for a Hub-based transaction translator.

All periodic transfers always begin at B-frame 0 (after SOF) and continue until the stored periodic transfers are complete. As an example of the microframe pipeline implemented in the embedded transaction translator, all periodic transfers that are tagged in EHCI to execute in H-frame 0 will be ready to execute on the bus in B-frame 0.

It is important to note that when programming the S-mask and C-masks in the EHCI data structures to schedule periodic transfers for the embedded transaction translator, the EHCI host controller driver must follow the same rules specified in EHCI for programming the S-mask and C-mask for downstream Hub-based transaction translators.

Once periodic transfers are exhausted, any stored asynchronous transfer are moved. Asynchronous transfers are opportunistic in that they shall execute whenever possible and their operation is not tied to H-frame and B-frame boundaries with the exception that an asynchronous transfer can not babble through the SOF (start of B-frame 0).

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 16-96](#) summarizes the conditions where handshakes are emulated from internal state instead of actual handshakes to HS split bus traffic.

Table 16-96. Emulated Handshakes

Condition	Emulate TT Response
Start-Split: All asynchronous buffers full	NAK
Start-Split: All periodic buffers full	ERR
Start-Split: Success for start of Async. Transaction	ACK
Start-Split: Start Periodic Transaction	No Handshake (Ok)
Complete-Split: Failed to find transaction in queue	Bus Time Out
Complete-Split: Transaction in Queue is Busy	NYET
Complete-Split: Transaction in Queue is Complete	[Actual Handshake from FS/LS device]

16.9.1.5.3 Asynchronous Transaction Scheduling and Buffer Management

The following *Universal Serial Bus Revision 2.0 Specification* items are implemented in the embedded transaction translator:

- USB 2.0–11.17.3
 - Sequencing is provided and a packet length estimator ensures no full-speed/low-speed packet babbles into SOF time.
- USB 2.0–11.17.4
 - Transaction tracking for 2 data pipes.
- USB 2.0–11.17.5
 - Clear_TT_Buffer capability provided

16.9.1.5.4 Periodic Transaction Scheduling and Buffer Management

The following *Universal Serial Bus Revision 2.0 Specification* items are implemented in the embedded transaction translator:

- USB 2.0–11.18.6.[1-2]
 - Abort of pending start-splits
 - EOF (and not started in microframes 6)
 - Idle for more than 4 microframes
 - Abort of pending complete-splits
 - EOF
 - Idle for more than 4 microframes

NOTE

There is no data schedule mechanism for these transactions other than the microframe pipeline. The embedded TT assumes the number of packets scheduled in a frame does not exceed the frame duration (1 msec) or else undefined behavior may result.

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

The co-existence of a device operational controller within the DR module has little effect on EHCI compatibility for host operation except as noted in this section.

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 μ sec interrupt for host mode. EHCI does not specify this interrupt, but it has been added for convenience and as a potential software time base. Note that the free running interrupt is shared with the device-mode start-of-frame interrupt. See [Section 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.

16.9.5.1 Frame Adjust Register

Starts of microframes are timed precisely to 125 μ sec using the transceiver clock as a reference clock. That is, 60 MHz transceiver clock for 8-bit physical interfaces and full-speed serial interfaces or 30 MHz transceiver clock for 16-bit physical interfaces.

16.9.6 Miscellaneous Variations from EHCI

The modules support multiple physical interfaces which can operate in different modes when the module is configured with the software programmable Physical Interface Modes. The control bits for selecting the PHY operating mode have been added to the PORTSC register providing a capability that is not defined by the EHCI specification.

16.9.6.1 Discovery

This section discusses port reset and port speed detection.

16.9.6.1.1 Port Reset

The port connect methods specified by EHCI require setting the port reset bit in the register for a duration of 10 msec. Due to the complexity required to support the attachment of devices that are not high speed there are counter already present in the design that can count the 10 msec reset pulse to alleviate the requirement of the software to measure this duration. Therefore, the basic connection is then summarized as the following:

- [Port Change Interrupt] Port connect change occurs to notify the host controller driver that a device has attached.
- Software shall write a '1' to the reset the device.
- Software shall write a '0' to the reset the device after 10 msec.
 - This step, which is necessary in a standard EHCI design, may be omitted with this implementation. Should the EHCI host controller driver attempt to write a '0' to the reset bit while a reset is in progress the write will simple be ignored and the reset will continue until completion.
- [Port Change Interrupt] Port enable change occurs to notify the host controller that the device in now operational and at this point the port speed has been determined.

16.9.6.1.2 Port Speed Detection

After the port change interrupt indicates that a port is enabled, the EHCI stack should determine the port speed. Unlike the EHCI implementation which will re-assign the port owner for any device that does not connect at High-Speed, this host controller supports direct attach of non-HS devices. Therefore, the following differences are important regarding port speed detection:

- Port owner is read-only and always reads 0.
- A 2-bit port speed indicator has been added to PORTSC to provide the current operating speed of the port to the host controller driver.

A 1-bit high-speed indicator has been added to PORTSC to signify that the port is in HS vs. FS/LS

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.

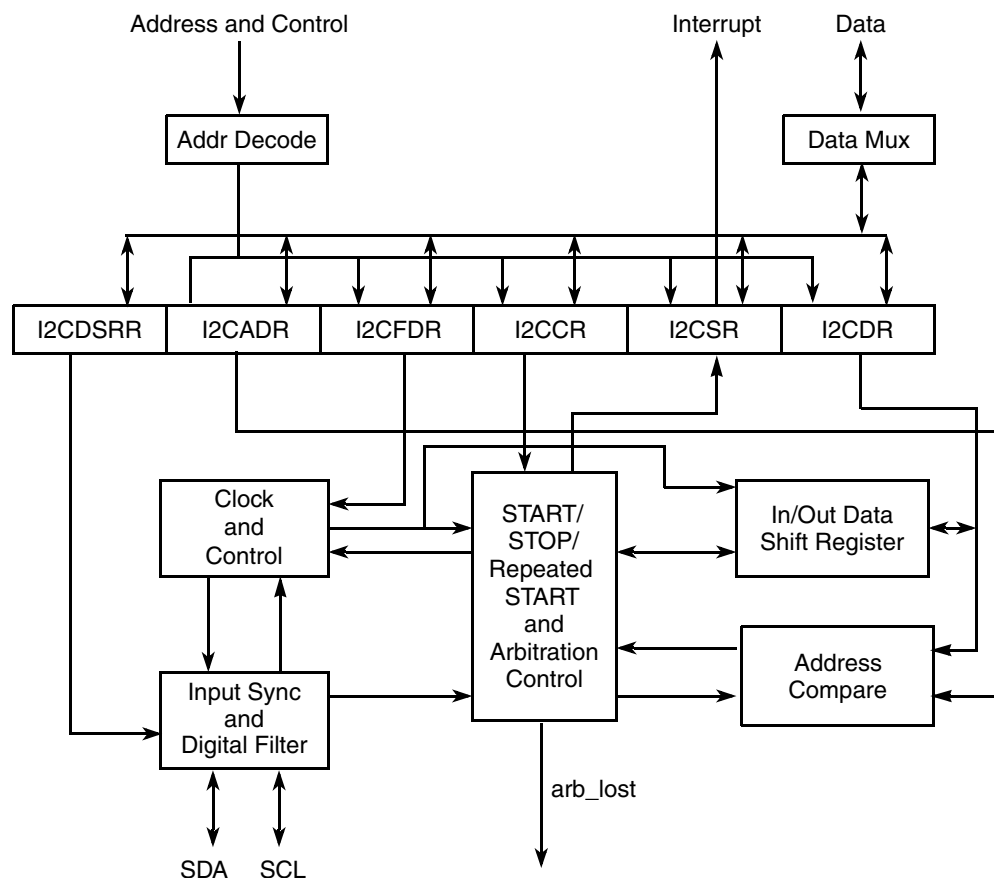


Figure 17-1. I²C Block Diagram

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.

MPC8309 has two instances of I²C controllers. I²C controller 1 is used for boot sequencing and I²C controller 2 is used for data communication.

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 17.4.5, “Boot Sequencer Mode.”](#)

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 Descriptions

Signal Name	Idle State	I/O	State Meaning
Serial Clock (SCL1, SCL2)	High	I	When the I ² C module is idle or acts as a slave, SCL _{<i>n</i>} defaults as an input. The unit uses SCL _{<i>n</i>} to synchronize incoming data on SDA _{<i>n</i>} . The bus is assumed to be busy when SCL _{<i>n</i>} is detected low.
		O	As a master, the I ² C module drives SCL _{<i>n</i>} along with SDA _{<i>n</i>} when transmitting. As a slave, the I ² C module drives SCL _{<i>n</i>} negates for data pacing.
Serial Data (SDA1, SDA2)	High	I	When the I ² C module is idle or in a receiving mode, SDA _{<i>n</i>} defaults as an input. The unit receives data from other I ² C devices on SDA _{<i>n</i>} . The bus is assumed to be busy when SDA _{<i>n</i>} is detected low.
		O	When writing as a master or slave, the I ² C module drives data on SDA _{<i>n</i>} synchronous to SCL _{<i>n</i>} .

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 *MPC8309 PowerQUICC II Pro Integrated Communications Processor Hardware Specification* 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 I ² C slave. SCL _n also performs as an output when the device is programmed as an I ² C master.
	O	As outputs for the bidirectional serial clock, these signals operate as described below.
		State Meaning
	I	As inputs for the bi-directional serial clock, these signals operate as described below.
State Meaning		Asserted/Negated—The I ² C unit uses this signal to synchronize incoming data on SDA _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 I ² C master or a slave).
	O	As outputs for the bi-directional serial data, these signals operate as described below.
		State Meaning
	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	I ² C Register	Access	Reset	Section/Page
I²C Controller 1—Block Base Address 0x0_3000 I²C Controller 2—Block Base Address 0x0_3100				
0x0_3000	I2C1ADR—I ² C1 address register	R/W	0x00	17.3.1.1/17-5
0x0_3004	I2C1FDR—I ² C1 frequency divider register	R/W	0x00	17.3.1.2/17-6
0x0_3008	I2C1CR—I ² C1 control register	R/W	0x00	17.3.1.3/17-7
0x0_300C	I2C1SR—I ² C1 status register	R/W	0x81	17.3.1.4/17-8
0x0_3010	I2C1DR—I ² C1 data register	R/W	0x00	17.3.1.5/17-9
0x0_3014	I2C1DFSRR—I ² C1 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—I ² C2 address register	R/W	0x00	17.3.1.1/17-5
0x0_3104	I2C2FDR—I ² C2 frequency divider register	R/W	0x00	17.3.1.2/17-6
0x0_3108	I2C2CR—I ² C2 control register	R/W	0x00	17.3.1.3/17-7
0x0_310C	I2C2SR—I ² C2 status register	R/W	0x81	17.3.1.4/17-8

Table 17-3. I²C Memory Map (continued)

Address	I ² C Register	Access	Reset	Section/Page
0x0_3110	I2C2DR—I ² C2 data register	R/W	0x00	17.3.1.5/17-9
0x0_3114	I2C2DFSRR—I ² C2 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_nDR).

17.3.1.1 I²C_n Address Register (I2C_nADR)

Figure 17-2 shows the I2C_nADR 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_nADR)

Table 17-4 describes the bit settings of I2C_nADR.

Table 17-4. I2C_nADR Field Descriptions

Bits	Name	Description
0–6	ADDR	Slave address. Contains the specific slave address that is used by the I ² C interface. Note that the default mode of the I ² C interface is slave mode for an address match. Note that an address match is one of the conditions that can cause 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 (I2CnFDR)

Figure 17-3 shows the bits of the I²C_n frequency divider register.

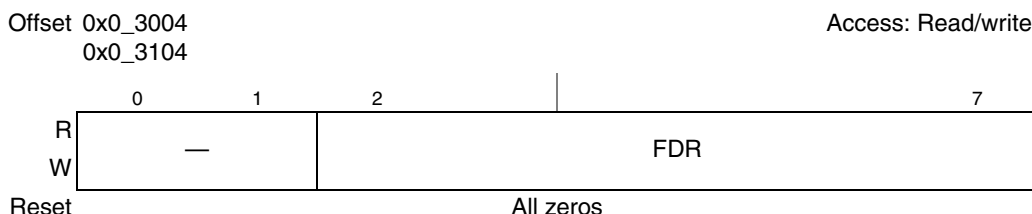


Figure 17-3. I²C_n Frequency Divider Register (I2CnFDR)

Table 17-5 describes the bit settings of I2CnFDR. It also maps I2CnFDR[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 1:1. Clock ratios of I²C₁ are controllable but clock ratio for I²C₂ is not and it is always 1:1 with CSB. 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	<p>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:</p> <table border="1"> <thead> <tr> <th>FDR</th> <th>Divider (Decimal)</th> <th>FDR</th> <th>Divider (Decimal)</th> <th>FDR</th> <th>Divider (Decimal)</th> </tr> </thead> <tbody> <tr><td>0x00</td><td>384</td><td>0x16</td><td>12288</td><td>0x2B</td><td>1024</td></tr> <tr><td>0x01</td><td>416</td><td>0x17</td><td>15360</td><td>0x2C</td><td>1280</td></tr> <tr><td>0x02</td><td>480</td><td>0x18</td><td>18432</td><td>0x2D</td><td>1536</td></tr> <tr><td>0x03</td><td>576</td><td>0x19</td><td>20480</td><td>0x2E</td><td>1792</td></tr> <tr><td>0x04</td><td>640</td><td>0x1A</td><td>24576</td><td>0x2F</td><td>2048</td></tr> <tr><td>0x05</td><td>704</td><td>0x1B</td><td>30720</td><td>0x30</td><td>2560</td></tr> <tr><td>0x06</td><td>832</td><td>0x1C</td><td>36864</td><td>0x31</td><td>3072</td></tr> <tr><td>0x07</td><td>1024</td><td>0x1D</td><td>40960</td><td>0x32</td><td>3584</td></tr> <tr><td>0x08</td><td>1152</td><td>0x1E</td><td>49152</td><td>0x33</td><td>4096</td></tr> <tr><td>0x09</td><td>1280</td><td>0x1F</td><td>61440</td><td>0x34</td><td>5120</td></tr> <tr><td>0x0A</td><td>1536</td><td>0x20</td><td>256</td><td>0x35</td><td>6144</td></tr> <tr><td>0x0B</td><td>1920</td><td>0x21</td><td>288</td><td>0x36</td><td>7168</td></tr> <tr><td>0x0C</td><td>2304</td><td>0x22</td><td>320</td><td>0x37</td><td>8192</td></tr> <tr><td>0x0D</td><td>2560</td><td>0x23</td><td>352</td><td>0x38</td><td>10240</td></tr> <tr><td>0x0E</td><td>3072</td><td>0x24</td><td>384</td><td>0x39</td><td>12288</td></tr> <tr><td>0x0F</td><td>3840</td><td>0x25</td><td>448</td><td>0x3A</td><td>14336</td></tr> <tr><td>0x10</td><td>4608</td><td>0x26</td><td>512</td><td>0x3B</td><td>16384</td></tr> <tr><td>0x11</td><td>5120</td><td>0x27</td><td>576</td><td>0x3C</td><td>20480</td></tr> <tr><td>0x12</td><td>6144</td><td>0x28</td><td>640</td><td>0x3D</td><td>24576</td></tr> <tr><td>0x13</td><td>7680</td><td>0x29</td><td>768</td><td>0x3E</td><td>28672</td></tr> <tr><td>0x14</td><td>9216</td><td>0x2A</td><td>896</td><td>0x3F</td><td>32768</td></tr> <tr><td>0x15</td><td>10240</td><td></td><td></td><td></td><td></td></tr> </tbody> </table> <p>Note: The values shown in the table are applicable only for the default value of DFSRR. Refer to AN2919.</p>	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				
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 (I2CnCR)

Figure 17-4 shows the I²C_n control register.

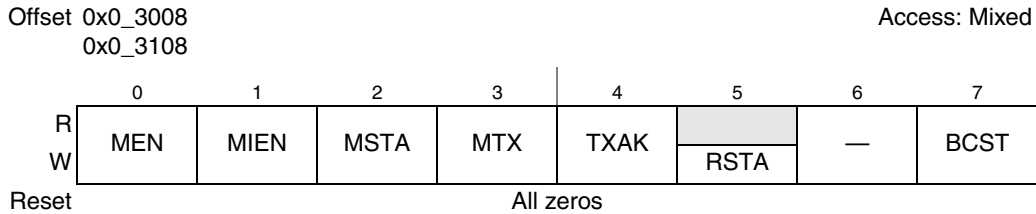


Figure 17-4. I²C_n Control Register (I2CnCR)

Table 17-6 describes the I2CnCR bit settings.

Table 17-6. I2CnCR Field Descriptions

Bits	Name	Description
0	MEN	Module enable. Controls the software reset of the I ² C module. 0 The module is reset and disabled. The interface is held in reset, but the registers can still be accessed. 1 The I ² C module is enabled. MEN must be set before any other control register bits have any effect. All I ² C registers for slave receive or master START can be initialized before setting this bit.
1	MIEN	Module interrupt enable 0 Interrupts from the I ² C module are disabled. This does not clear any pending interrupt conditions. 1 Interrupts from the I ² C module are enabled. An interrupt occurs provided I2CnSR[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 I2CnSR[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 I ² C module is configured as a receiver, not a transmitter. It also does not apply to address cycles; when the device is addressed as a slave, an acknowledge is always sent. 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.

Table 17-6. I2CnCR Field Descriptions (continued)

Bits	Name	Description
6	—	Reserved, should be cleared
7	BCST	Broadcast 0 Disables the broadcast accept capability 1 Enables the I ² C to accept broadcast messages at address zero

17.3.1.4 I²Cn Status Register (I2CnSR)

I2CSR is shown in [Figure 17-5](#).

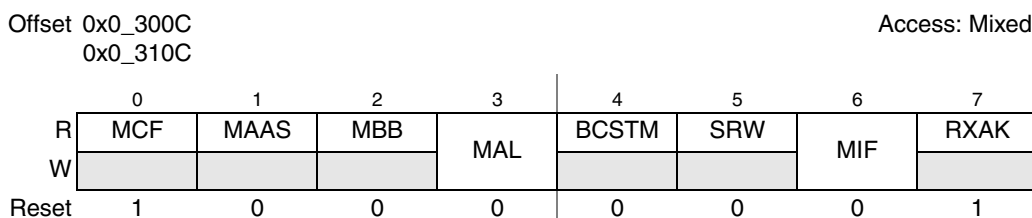


Figure 17-5. I²Cn Status Register (I2CnSR)

[Table 17-7](#) describes the bit settings of the I2CnSR.

Table 17-7. I2CnSR Field Descriptions

Bits	Name	Description
0	MCF	Data transfer. When one byte of data is transferred, the bit is cleared. It is set by the falling edge of the 9th clock of a byte transfer. 0 Byte transfer in progress. MCF is cleared under the following conditions: <ul style="list-style-type: none"> When I2CnDR is read in receive mode or when I2CnDR 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 I2CnADR matches the calling address or when the calling address is the broadcast address and broadcast mode is enabled (I2CnCR[BCST] is set), this bit is set. The processor is interrupted if I2CnCR[MIE] is set. Next, the processor must check the SRW bit and set I2CnCR[MTX] accordingly. Writing to the I2CnCR automatically clears this bit. 0 Not addressed as a slave 1 Addressed as a slave
2	MBB	Bus busy. Indicates the status of the bus. When a START condition is detected, MBB is set. If a STOP condition is detected, it is cleared. 0 I ² C bus is idle 1 I ² C bus is busy
3	MAL	Arbitration lost. Automatically set when the arbitration procedure is lost. Note that the device does not automatically retry a failed transfer attempt. 0 Arbitration is not lost. Can only be cleared by software 1 Arbitration is lost
4	BCSTM	Broadcast match. Writing to the I2CnCR automatically clears this bit. 0 There has not been a broadcast match. 1 The calling address matches with the broadcast address and broadcast mode is enabled. This is also set if this I ² C drives an address of all 0s.

Table 17-7. I2CnSR Field Descriptions (continued)

Bits	Name	Description
5	SRW	Slave read/write. When MAAS is set, SRW indicates the value of the R/W command bit of the calling address, which is sent from the master. 0 Slave receive, master writing to slave 1 Slave transmit, master reading from slave. This bit is valid only when both of the following conditions are true: <ul style="list-style-type: none"> • 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.
6	MIF	Module interrupt. The MIF bit is set when an interrupt is pending, causing a processor interrupt request (provided I2CnCR[MIEN] is set). 0 No interrupt is pending. Can be cleared only by software. 1 Interrupt is pending. MIF is set when one of the following events occurs: <ul style="list-style-type: none"> • One byte of data is transferred (set at the falling edge of the 9th clock). • The value in I2CnADR 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 I²Cn Data Register (I2CnDR)

The I2Cn data register is shown in Figure 17-6.

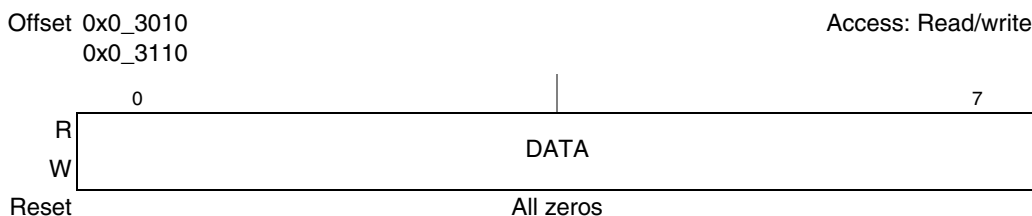


Figure 17-6. I²Cn Data Register (I2CnDR)

Table 17-8 shows the bit descriptions for I2CnDR.

Table 17-8. I2CnDR Field Descriptions

Bits	Name	Description
0–7	DATA	Transmission starts when an address and the R/W bit are written to the data register and the I ² C interface performs as the master. A data transfer is initiated when data is written to the I2CnDR. The most-significant bit is sent first in both cases. In master receive mode, reading the data register allows the read to occur, but also allows the I ² C module to receive the next byte of data on the I ² C interface. In slave mode, the same function is available after it is addressed. Note that in both master receive and slave receive modes, the very first read is always a dummy read.

17.3.1.6 Digital Filter Sampling Rate Register (I2CnDFSRR)

I2CnDFSRR is shown in Figure 17-7.

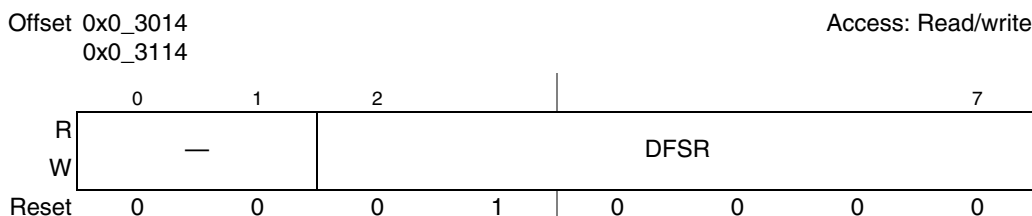


Figure 17-7. I²Cn Digital Filter Sampling Rate Register (I2CnDFSRR)

Table 17-9 shows the I2CnDFSRR field descriptions.

Table 17-9. I2CnDFSRR Field Descriptions

Bits	Name	Description
0–1	—	Reserved, should be cleared
2–7	DFSRR	Digital filter sampling rate. To assist in filtering out signal noise, the sample rate is programmed. DFSRR is used to prescale the frequency at which the digital filter takes samples from the I ² C bus. The resulting sampling rate is calculated by dividing the platform frequency by the non-zero value of DFSRR. If I2CnDFSRR is cleared, the I ² C 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.

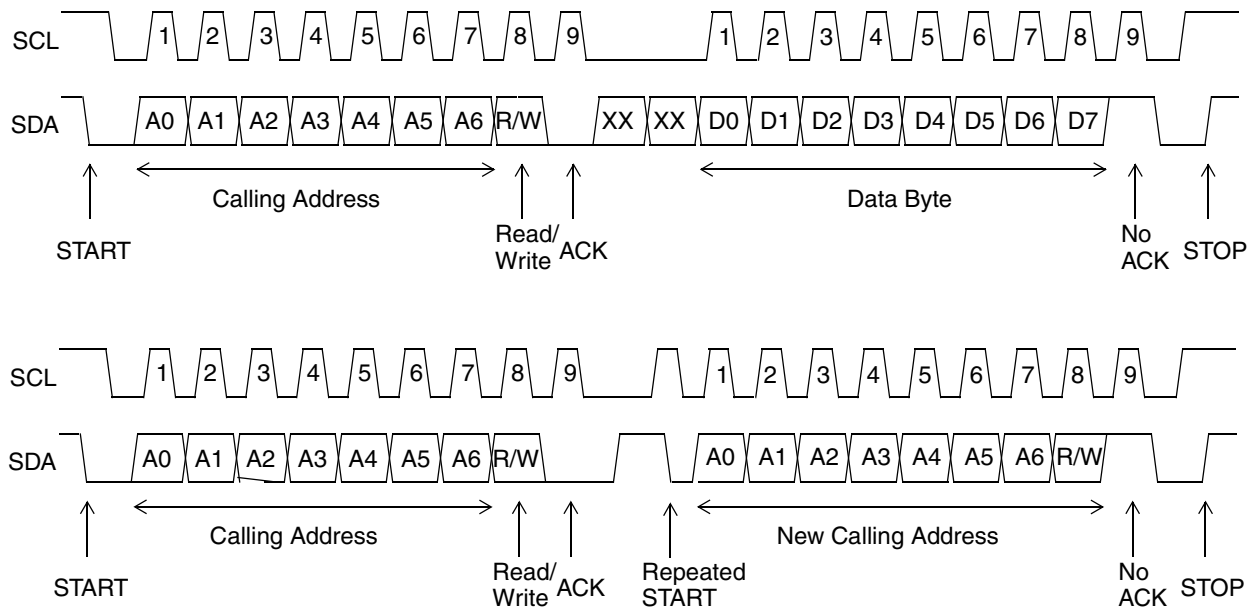


Figure 17-8. I²C Interface Transaction Protocol

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 I2CnCR[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/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/W bit sent by the calling master.

The I²C module responds to a general call (broadcast) command when I2CnCR[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 I2CnDR with I2CnCR[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 I2CnCR[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 I2CnSR
- Whether the module has been addressed as a slave, to update I2CnSR 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 I2CnSR[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 I2CnSR[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 (RCWHR). 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 4.3.2.2.2, “Boot Sequencer Configuration,”](#) the default value for BOOTSEQ is 0b00, which corresponds to the I²C boot sequencer being disabled at power-up.

If the boot sequencer is enabled for standard I²C addressing mode, the I²C interface initiates the following sequence during reset:

1. Generate RESET sequence (START then 9 SCL cycles) to the EEPROM twice. This clears any transactions that may have been in progress prior to the reset.
2. Generate START.
3. Transmit 0xA0 which is the 7-bit calling address (0b101_0000) with a write command appended (0 as the least significant bit).
4. Transmit 0x00 which is the high-order starting address.
5. Transmit 0x00 which is the low-order starting address.
6. Generate a repeated START.
7. Transmit 0xA1 which is the 7-bit calling address (0b101_0000) with a read command appended (1 as the least significant bit).
8. Receive data continuously from the EEPROM until continue (CONT) bit is cleared and the CRC check is executed.

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 is 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 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.

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 4.3.2.2.2, “Boot Sequencer Configuration.”](#)) In this mode, only one EEPROM device can be used and the maximum number of registers is limited by the size of the EEPROM.

1. Generate RESET sequence (START then 9 SCL cycles) to the EEPROM twice. This clears any transactions that may have been in progress prior to the reset.
2. Generate START.
3. Transmit 0xA0 which is the 7-bit calling address (0b101_0000) with a write command appended (0 as the least significant bit).
4. Transmit 0x00 which is the high-order starting address.
5. Transmit 0x00 which is the low-order starting address.
6. Generate a repeated START.
7. Transmit 0xA1 which is the 7-bit calling address (0b101_0000) with a read command appended (1 as the least significant bit).
8. Receive data continuously from the EEPROM until the CONT bit is cleared and the CRC check is executed. See [Section 17.4.5.3, “EEPROM Data Format,”](#) for more information.

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 4.3.2.2.2, “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 I2CnCR[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.

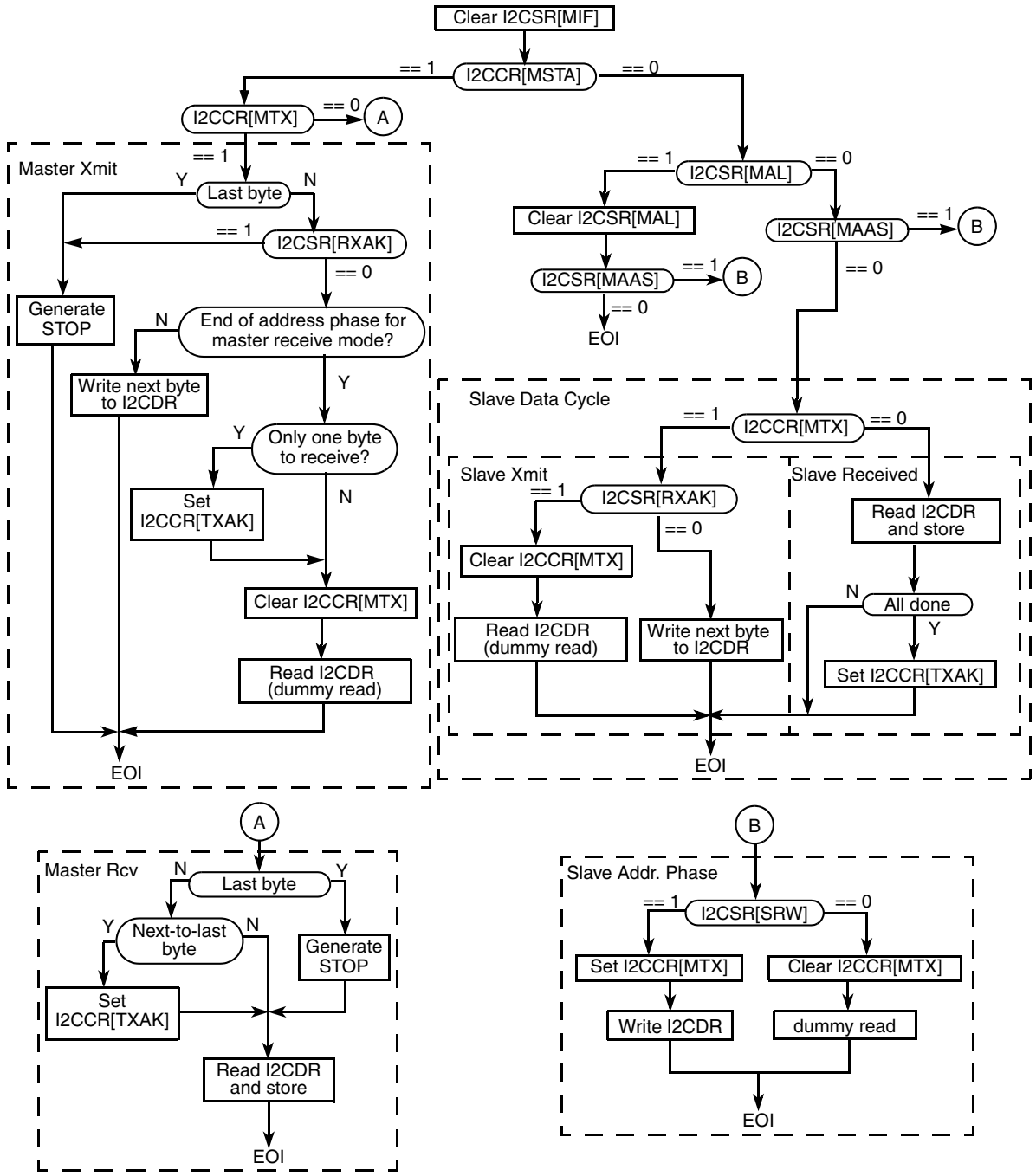


Figure 17-11. Example I²C Interrupt Service Routine Flowchart

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 I2CnFDR[FDR] and select the required division ratio to obtain the SCLn frequency from the CSB (platform) clock.
3. Update I2CnADR to define the slave address for this device.
4. Modify I2CnCR to select master/slave mode, transmit/receive mode, and interrupt-enable or disable.
5. Set the I2CnCR[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 (I2CnSR[MBB] = 0) before switching to master mode.
2. Select master mode (set I2CnCR[MSTA]) to transmit serial data and select transmit mode (set I2CnCR[MTX]) for the address cycle.
3. Write the slave address being called into I2CnDR. The data written to I2CnDR[0–6] comprises the slave calling address. I2CnCR[MTX] indicates the direction of transfer (transmit/receive) required from the slave.

The scenario above assumes that the I²C interrupt bit (I2CnSR[MIF]) is cleared. If MIF is set at any time, an I²C interrupt is generated (provided interrupt reporting is enabled with I2CnCR[MIEN] = 1).

17.5.4 Post-Transfer Software Response

Transmission or reception of a byte automatically sets the data transferring bit (I2CnSR[MCF]), which indicates that one byte has been transferred. The I²C interrupt bit (I2CnSR[MIF]) is also set and an interrupt is generated to the processor if the interrupt function is enabled during the initialization sequence (I2CnCR[MIEN] is set). In the interrupt handler, software must take the following steps:

1. Clear I2CnSR[MIF]
2. Read the I2CnDR in receive mode or write to I2CnDR in transmit mode. Note that this causes I2CnSR[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, I2CnCR[MTX] must be toggled at this stage (see [Figure 17-11](#)).

If the interrupt function is disabled, software can service the I2CnDR in the main program by monitoring I2CnSR[MIF]. In this case, I2CnSR[MIF] must be polled rather than I2CnSR[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 I2CnSR[MIF] (or any other I2CnSR bits), software delays may be needed to give the I²C signals sufficient time to settle.

During slave-mode address cycles (I2CnSR[MAAS] is set), I2CnSR[SRW] should be read to determine the direction of the subsequent transfer and I2CnCR[MTX] should be programmed accordingly. For slave-mode data cycles (MAAS is cleared), I2CnSR[SRW] is not valid and I2CnCR[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 (I2CnCR[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 I2CnCR[TXAK] is set). Before the interrupt service routine reads the last byte of data, a STOP condition must first be generated.

17.5.6 Generation of Repeated START

At the end of a data transfer, if the master still wants to communicate on the bus, it can generate another START condition followed by another slave address without first generating a STOP condition. This is accomplished by setting I2CnCR[RSTA].

17.5.7 Generation of SCLn When SDAn is Negated

It is sometimes necessary to force the I²C module to become the I²C bus master out of reset and drive SCLn (even though SDAn 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, SDAn 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 SCLn so that the device driving SDAn can finish its transaction:

1. Disable the I²C module and set the master bit by setting I2CnCR to 0x20.
2. Enable the I²C module by setting I2CnCR to 0xA0.
3. Read I2CnDR.
4. Return the I²C module to slave mode by setting I2CnCR 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 I2CnSR[MAAS] is set, software should set the transmit/receive mode select bit (I2CnCR[MTX]) according to the R \bar{W} command bit (I2CnSR[SRW]). Writing to I2CnCR 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 I2CnDR for slave transmits or dummy reading from I2CnDR in slave-receive mode. The slave negates SCLn between byte transfers. SCLn is released when I2CnDR is accessed in the required mode.

17.5.8.1 Slave Transmitter and Received Acknowledge

In the slave transmitter routine, the received acknowledge bit (I2CnSR[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 (I2CnSR[RXAK] is set), the slave transmitter interrupt routine must clear I2CnCR[MTX] to switch the slave from transmitter to receiver mode. A dummy read of I2CnDR 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:

- I2CnSR[MAL] is set
- I2CnCR[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 I2CnSR[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 (DUARTs) of the device. It describes the functional operation, the DUART initialization sequence, and the programming details for the DUART registers and features.

18.1 Overview

MPC8309 support two DUARTs each. A 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

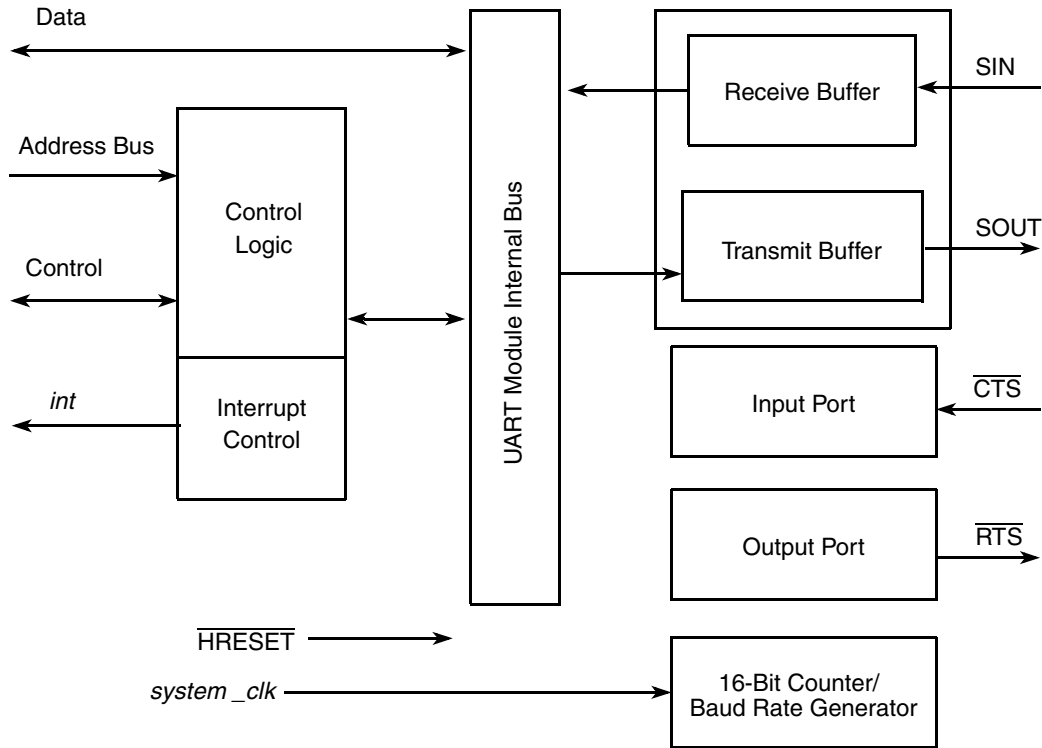


Figure 18-1. UART Block Diagram

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{CTS}) and ready-to-send (\overline{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
$\overline{\text{UART1_CTS}}[1]$	I	2	1	Clear to send UART1 and UART2
$\overline{\text{UART1_RTS}}[1]$	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, 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, <i>rxcnt</i> , 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, UART2 are set (mark condition) when the transmitter is disabled, idle, or operating in the local loopback mode. Data is shifted out on these signals, with the least significant bit transmitted first.	
		State Meaning	Asserted/Negated—Represents the data transmitted on the respective UART interface.
		Timing	Assertion/Negation—An internal logic sample signal, <i>rxcnt</i> , uses the frequency of the baud-rate generator to update and drive the data on SOUT.
UART1_CTS1	I	Clear to send. Connected to the respective \overline{RTS} outputs of the 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.
UART1_RTS1	O	Request to send. Can be programmed to be negated and asserted by either the receiver or transmitter. When connected to the \overline{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

For MPC8309, there are four complete sets of DUART registers (one each for UART1, UART2, UART3, and UART4). The four UARTs are identical, except that the registers for:

- UART1 are located at offsets 0x0_4500 (local)
- UART2 are located at offsets 0x0_4600 (local)
- UART3 are located at offsets 0x0_4900 (local)
- UART4 are located at offsets 0x0_4A00 (local)

Throughout this chapter, the registers are described by a singular acronym: for example, LCR represents the line control register for either UART1, UART2, UART3, or UART4.

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 18.3.1.8, “Line Control Registers \(ULCR1 and ULCR2, ULCR3 and ULCR4\),”](#) 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 0x0_4900–4910	URBR—ULCR[DLAB] = 0 UART1 receiver buffer register	R	0x00	18.3.1.1/18-6
	U THR—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
	UA FR—ULCR[DLAB] = 1 UART1 alternate function register	R/W	0x00	18.3.1.7/18-11
0x0_4503	ULCR—ULCR[DLAB] = x UART1 line control register	R/W	0x00	18.3.1.8/18-12
0x0_4504	UMCR—ULCR[DLAB] = x UART1 MODEM control register	R/W	0x00	18.3.1.9/18-14
0x0_4505	ULSR—ULCR[DLAB] = x UART1 line status register	R	0x60	18.3.1.10/18-15
0x0_4506	UMSR—ULCR[DLAB] = x UART1 MODEM status register	R	0x00	18.3.1.11/18-16
0x0_4507	USCR—ULCR[DLAB] = x UART1 scratch register	R/W	0x00	18.3.1.12/18-17
0x0_4510	UDSR—ULCR[DLAB] = x UART1 DMA status register	R	0x01	18.3.1.13/18-17
0x0_4600 0x0_4A00–4A10	URBR—ULCR[DLAB] = 0 UART2 receiver buffer register	R	0x00	18.3.1.1/18-6
	U THR—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
	UA FR—ULCR[DLAB] = 1 UART2 alternate function register	R/W	0x00	18.3.1.7/18-11
0x0_4603	ULCR—ULCR[DLAB] = x UART2 line control register	R/W	0x00	18.3.1.8/18-12
0x0_4604	UMCR—ULCR[DLAB] = x UART2 MODEM control register	R/W	0x00	18.3.1.9/18-14
0x0_4605	ULSR—ULCR[DLAB] = x UART2 line status register	R	0x60	18.3.1.10/18-15
0x0_4606	UMSR—ULCR[DLAB] = x UART2 MODEM status register	R	0x00	18.3.1.11/18-16
0x0_4607	USCR—ULCR[DLAB] = x UART2 scratch register	R/W	0x00	18.3.1.12/18-17
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 DUART1 and DUART2 registers.

18.3.1.1 Receiver Buffer Registers (URBR1 and URBR2, URBR3 and URBR4)

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 18.3.1.10, “Line Status Registers \(ULSR1 and ULSR2, ULSR3 and ULSR4\).”](#) Figure 18-2 shows the receiver buffer registers. Note that these registers have same offset as the UTHR.

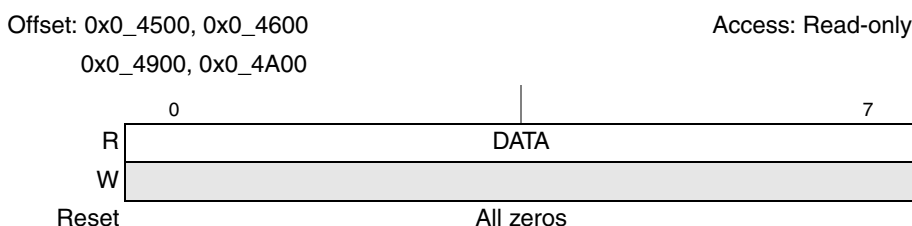


Figure 18-2. Receiver Buffer Registers (URBR1 and URBR2)

Table 18-4 describes URBR.

Table 18-4. URBR Field Descriptions

Bits	Name	Description
0–7	DATA	Data received from the transmitter on the UART bus [read only]

18.3.1.2 Transmitter Holding Registers (UTHR1 and UTHR2, UTHR3 and UTHR4)

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 UTHR.

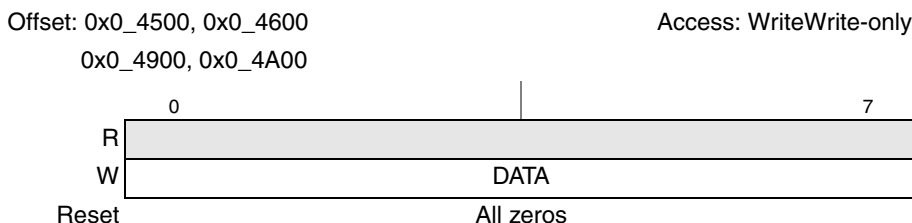


Figure 18-3. Transmitter Holding Registers (UTHR1 and UTHR2, UTHR3 and UTHR4)

Table 18-5 describes the UTHR.

Table 18-5. UTHR Field Descriptions

Bits	Name	Description
0–7	DATA	Data that is written to UTHR [Write only]

18.3.1.3 Divisor Most and Least Significant Byte Registers (UDMB and UDLB)

UDLB is concatenated with the divisor most significant byte register (UDMB) to create the divisor used to divide the input clock into the DUART. The output frequency of the baud generator is 16 times the baud rate; therefore, the desired baud rate = platform clock frequency \div (16 \times [UDMB||UDLB]). Equivalently, [UDMB||UDLB:0b0000] = platform clock frequency/desired baud rate. Baud rates that can be generated by specific input clock frequencies are shown in Table 18-8.

Figure 18-4 shows the bits in the UDMBs.

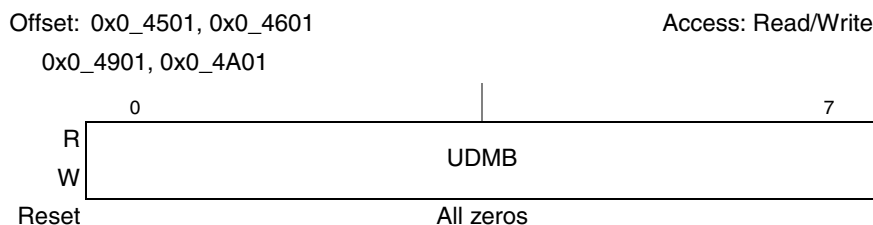


Figure 18-4. Divisor Most Significant Byte Registers (UDMB1 and UDMB2, UDMB3 and UDMB4)

Table 18-6 describes the UDMB.

Table 18-6. UDMB Field Descriptions

Bits	Name	Description
0–7	UDMB	Divisor most significant byte

Figure 18-5 shows the bits in the UDLBs.

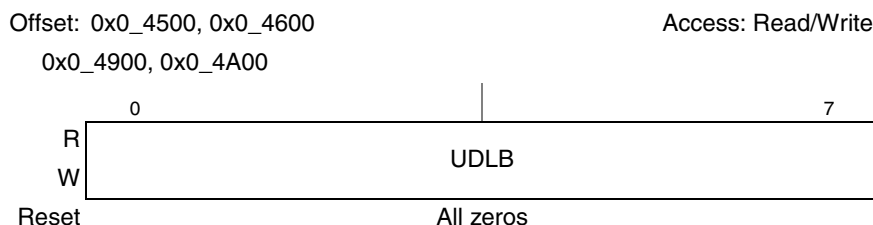


Figure 18-5. Divisor Least Significant Byte Registers (UDLB1 and UDLB2, UDLB3 and UDLB4)

Table 18-7 describes the UDLB.

Table 18-7. UDLB Field Descriptions

Bits	Name	Description
0–7	UDLB	Divisor least significant byte. This is concatenated with UDMB.

Table 18-8 shows baud rate for a variety of input clock frequencies.

Table 18-8. Baud Rate Examples

Baud Rate (Decimal)	Divisor		Input Clock (System Clock) Frequency (MHz)	Percent Error (Decimal)
	Decimal	Hex		
9,600	866	362	133	0.013
19,200	433	1B1	133	0.013
38,400	216	D8	133	0.218
56,000	148	94	133	0.300
128,000	65	41	133	0.090
256,000	32	20	133	1.471

To get the percent error value, the following three steps are taken:

1. The input clock frequency (ICF) is divided by the actual frequency input (AFI) to get the correct divisor value (ICF/AFI, where AFI = baud rate × 16 × divisor).
2. The divisor value is subtracted from 1.
3. The result from the step two is multiplied by 100 to calculate the final percent error. The result is calculated in absolute value (no negative numbers).

These steps can be described with the following equation:

$$\text{Percent error value} = (1 - \text{AFI/ICF}) \times 100$$

18.3.1.4 Interrupt Enable Registers (UIER1 and UIER2, UIER3 and UIER4)

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.

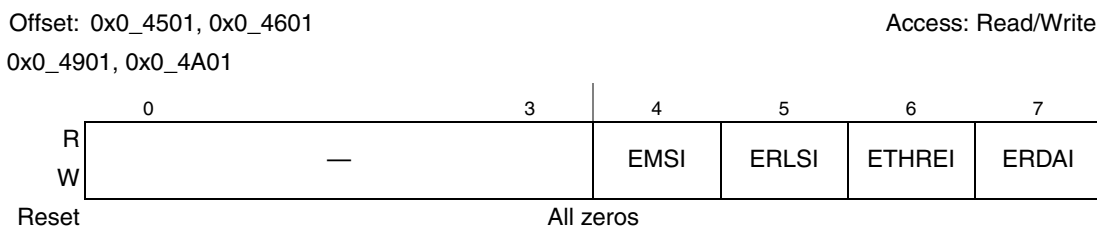


Figure 18-6. Interrupt Enable Registers (UIER1 and UIER2, UIER3 and UIER4)

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, UIIR3 and UIIR4)

The UIIRs indicate when an interrupt is pending from the corresponding UART and what type of interrupt is active. They also indicate if the FIFOs are enabled.

The DUART prioritizes interrupts into four levels and records these in the corresponding UIIR. The four levels of interrupt conditions in order of priority are as follows:

1. Receiver line status
2. Received data ready/character time-out
3. Transmitter holding register empty
4. MODEM status

See Table 18-11 for more details.

When the UIIR is read, the associated DUART serial channel freezes all interrupts and indicates the highest priority pending interrupt. While this read transaction is occurring, the associated DUART serial channel records new interrupts, but does not change the contents of UIIR until the read access is complete.

Figure 18-7 shows the bits in the UIIR.

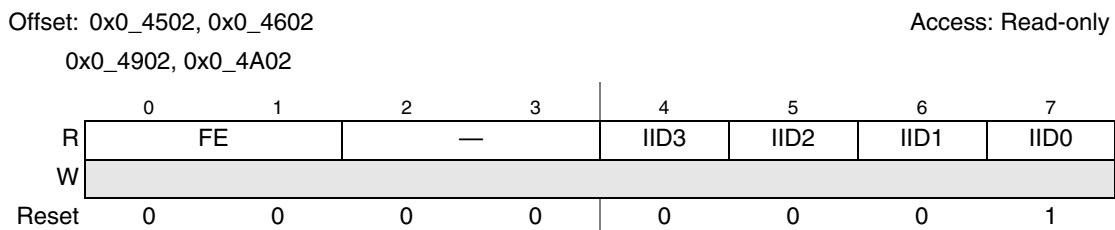


Figure 18-7. Interrupt ID Registers (UIIR1 and UIIR2, UIIR3 and UIIR4)

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, UFCR3 and UFCR4)

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.

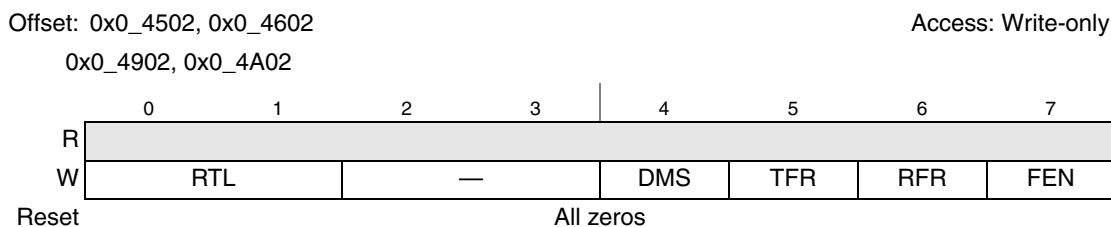


Figure 18-8. FIFO Control Registers (UFCR1 and UFCR2, UFCR3 and UFCR4)

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 Alternate Function Registers (UAFR1 and UAFR2, UAFR3 and UAFR4)

The UAFRs, shown in [Figure 18-9](#), allow software to write to both UART1 and UART2 registers, and UART3 and UART4 registers simultaneously with the same write operation. The UAFRs also provide a means for the device's performance monitor to track the baud clock.

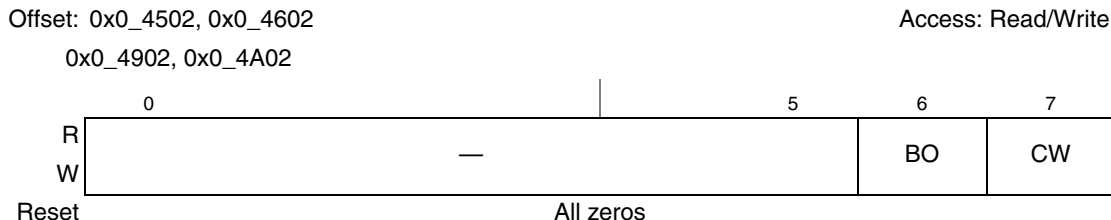


Figure 18-9. Alternate Function Register (UAFR)

Table 18-13 describes UAFR fields.

Table 18-13. 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.

18.3.1.8 Line Control Registers (ULCR1 and ULCR2, ULCR3 and ULCR4)

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 18-15. 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 18-10 shows the bits in the ULCRs.

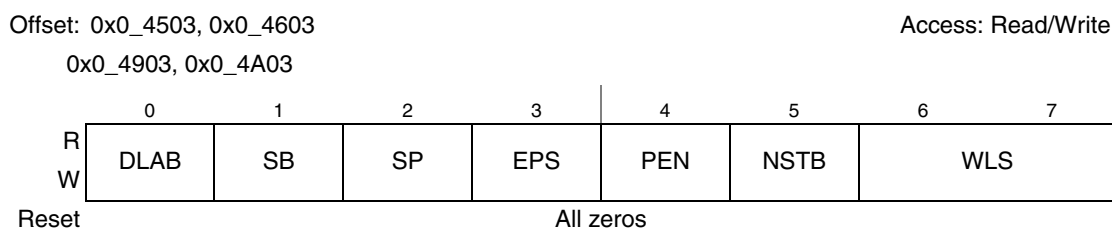


Figure 18-10. Line Control Register (ULCR1 and ULCR2, ULCR3 and ULCR4)

Table 18-14 describes the ULCR fields.

Table 18-14. 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 18-15 . 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 18-15. 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

18.3.1.9 MODEM Control Registers (UMCR1 and UMCR2, UMCR3 and UMCR4)

The UMCRs, shown in [Figure 18-11](#), control the interface with the external peripheral device on the UART bus.

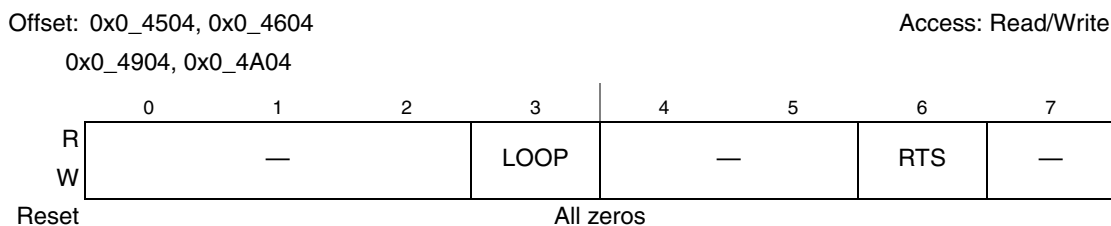


Figure 18-11. Modem Control Register (UMCR1 and UMCR2, UMCR3 and UMCR4)

[Table 18-16](#) describes the UMCR fields.

Table 18-16. UMCR Field Descriptions

Bits	Name	Description
0–2	—	Reserved, should be cleared
3	LOOP	Local loopback mode 0 Normal operation. 1 Functionally, the data written to UTHR can be read from URBR of the same UART, and UMCR[RTS] is tied to UMSR[CTS].
4–5	—	Reserved
6	RTS	Ready to send 0 Negates corresponding <u>UART1_RTS</u> output. 1 Assert corresponding <u>UART1_RTS</u> output. Informs external MODEM or peripheral that the UART is ready for sending/receiving data.
7	—	Reserved

18.3.1.10 Line Status Registers (ULSR1 and ULSR2, ULSR3 and ULSR4)

The ULSRs, shown in [Figure 18-12](#), monitor the status of the data transfer on the UART buses. To isolate the status bits from the proper character received through the UART bus, software should read the ULSR and then the URBR.

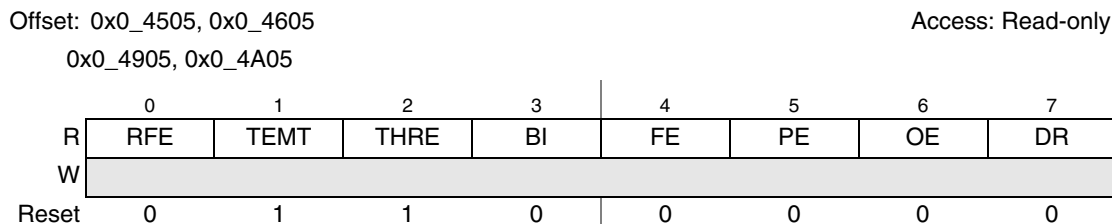


Figure 18-12. Line Status Register (ULSR1 and ULSR2, ULSR3 and ULSR4)

[Table 18-17](#) describes the ULSR fields.

Table 18-17. 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.

Table 18-17. ULSR Field Descriptions (continued)

Bits	Name	Description
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.
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.

18.3.1.11 MODEM Status Registers (UMSR1 and UMSR2, UMSR3 and UMSR4)

The UMSRs, shown in [Figure 18-13](#), track the status of the MODEM (or external peripheral device) $\overline{\text{CTS}}$, set for the corresponding UART.

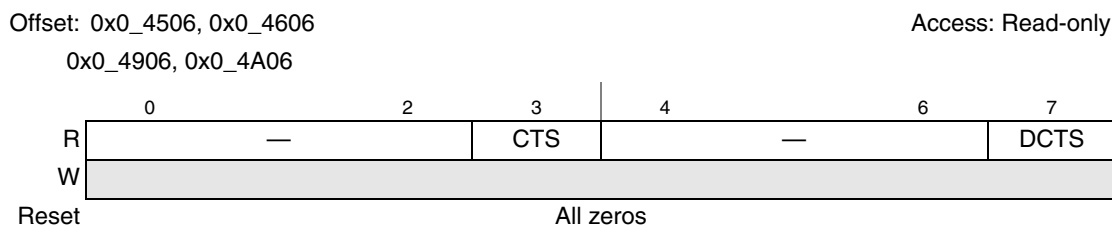


Figure 18-13. Modem Status Register (UMSR1 and UMSR2, UMSR3 and UMSR4)

[Table 18-18](#) describes UMSR fields.

Table 18-18. 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 $\overline{\text{CTS}}$ is negated. 1 $\overline{\text{CTS}}$ 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 $\overline{\text{CTS}}$ signal since the last read of UMSR[CTS]. 1 $\overline{\text{CTS}}$ changed since the last read of UMSR[CTS]. Causes an interrupt if UIER[EMSI] is set to detect this condition.

18.3.1.12 Scratch Registers (USCR1 and USCR2, USCR3 and USCR4)

USCR, shown in [Figure 18-14](#), are for debugging software or the DUART hardware. The USCRs do not affect the operation of the DUART.

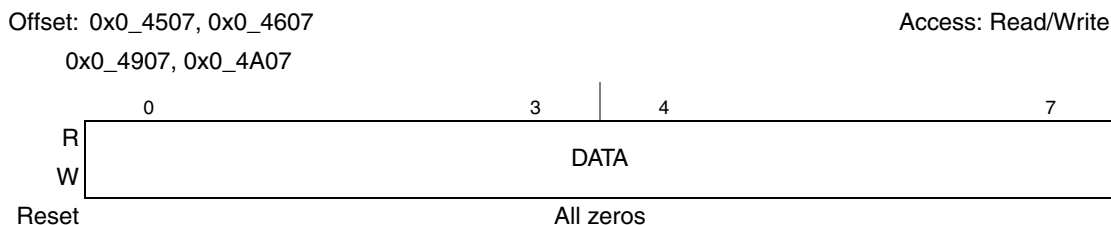


Figure 18-14. Scratch Register (USCR)

[Table 18-19](#) describes USCR fields.

Table 18-19. USCR Field Descriptions

Bits	Name	Description
0–7	DATA	Data

18.3.1.13 DMA Status Registers (UDSR1 and UDSR2, UDSR3 and UDSR4)

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.

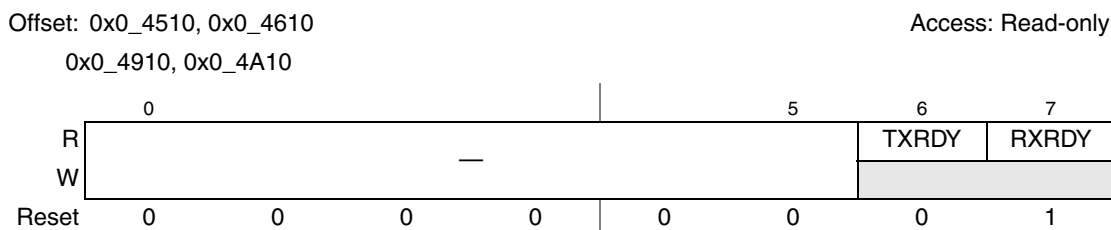


Figure 18-15. DMA Status Register (UDSR)

[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 in Table 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.

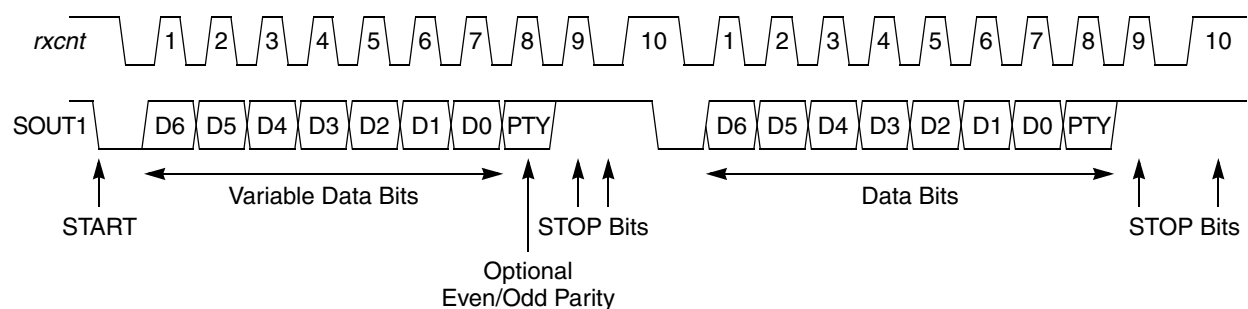
The transmitter accepts parallel data with a write access to UTHR. In FIFO mode, the data is placed directly into an internal transmitter shift register, or into the transmitter FIFO—see [Section 18.4.5, “FIFO Mode.”](#) The transmitting registers convert the data to a serial bit stream by inserting 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 loopback mode, different errors, and FIFO mode.

18.4.1.1 START Bit

A write to UTHR generates a START bit on the SOUT signal. [Figure 18-16](#) 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 18.3.1.8, “Line Control Registers \(ULCR1 and ULCR2, ULCR3 and ULCR4\).”](#) 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 18.3.1.10, “Line Status Registers \(ULSR1 and ULSR2, ULSR3 and ULSR4\).”](#)).

18.4.1.4 STOP Bit

The transmitter device ends the write transfer by generating a STOP bit. The STOP bit is always high. The user can program the length of the STOP bit(s) in the ULCR. Both the receiver and transmitter STOP bit length must agree before attempting to transfer data. A framing error can occur if an invalid STOP bit is detected.

18.4.2 Baud-Rate Generator Logic

Each UART contains an independent programmable baud-rate generator, that is capable of taking the system clock input and dividing the input by any divisor from 1 to $2^{16} - 1$. The baud rate is defined as the number of bits per second that can be sent over the UART bus. The formula for calculating baud rate is as follows:

$$\text{Baud rate} = (1/16) \times (\text{system clock frequency}/\text{divisor value})$$

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 Loopback Mode

Local loopback mode is provided for diagnostic testing. The data written to UTHR can be read from the receiver buffer register (URBR) of the same UART. In this mode, the MODEM control register UMCR[RTS] is internally tied to the MODEM status register UMSR[CTS]. The transmitter SOUT is set to a logic 1 and the receiver SIN is disconnected. The output of the transmitter shift register is looped back into the receiver shift register input. The $\overline{\text{CTS}}$ (input signal) is disconnected, $\overline{\text{RTS}}$ is internally connected to $\overline{\text{CTS}}$, and the $\overline{\text{RTS}}$ (output signal) becomes inactive. In this diagnostic mode, data that is transmitted is immediately received. In local loopback mode the transmit and receive data paths of the DUART can be verified. Note that in local loopback mode, the transmit/receive interrupts are fully operational and can be controlled by the interrupt enable register (UIER).

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 \(UIER1 and UIER2, UIER3 and UIER4\)”](#).

`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[IIDn]`) is cleared when `URBR` is read. See [Section 18.3.1.5, “Interrupt ID Registers \(UIIR1 and UIIR2, UIIR3 and UIIR4\)”](#).

`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 \(USDR1 and USDR2, USDR3 and USDR4\)”](#) 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 7 (`UIIR[IID0]`), is cleared. `UIER` is used to mask specific interrupt types. See [Section 18.3.1.4, “Interrupt Enable Registers \(UIER1 and UIER2, UIER3 and UIER4\)”](#).

When the interrupts are disabled in `UIER`, polling software can not use `UIIR[IID0]` to determine whether the UART is ready for service. Software must monitor the appropriate `ULSR` and `UMSR` bits. `UIIR[IID0]` can be used for polling if the interrupts are enabled in `UIER`.

18.5 DUART Initialization/Application Information

The following requirements must be met for DUART accesses:

- All DUART registers must be mapped to a cache-inhibited and guarded area. (That is, the WIMG setting in the MMU needs to be 0b01x1.)
- All DUART registers 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 enabled.

Chapter 19

Serial Peripheral Interface

19.1 Overview

The serial peripheral interface (SPI) allows the device to exchange data between other PowerQUICC® family chips, the MC68360, MC68302, M68HC11, and M68HC05 microcontroller families, and other family devices. The SPI can be used to communicate with peripheral devices such as EEPROMs, real-time clocks, A/D converters, and ISDN devices.

The SPI is a full-duplex, synchronous, character-oriented channel that supports a four-wire interface (receive, transmit, clock, and slave select). The SPI block consists of transmitter and receiver sections, an independent baud-rate generator, and a control unit. The transmitter and receiver sections use the same clock, which is derived from the SPI baud rate generator in master mode or externally in slave mode. During an SPI transfer, data is sent and received simultaneously.

The SPI receiver and transmitter are double-buffered, as shown in [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](#).

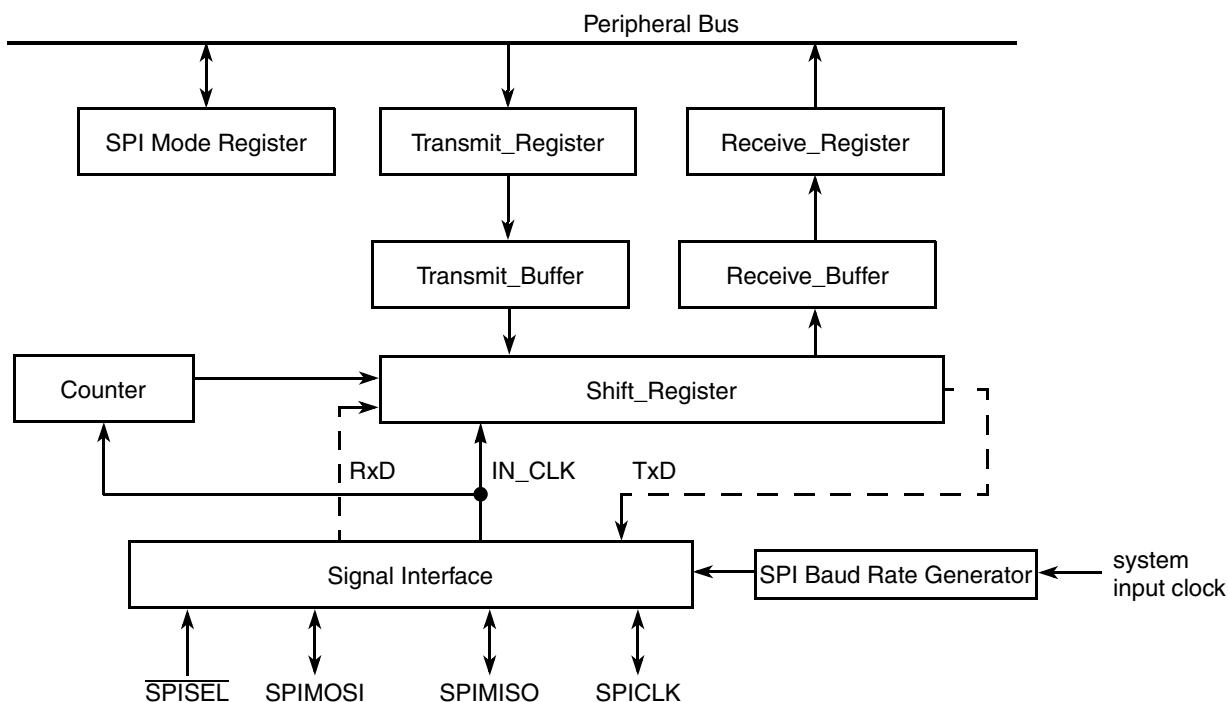


Figure 19-1. SPI Block Diagram

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 (input clock rate/4) in master mode; (input clock rate/2) in slave mode
- Independent programmable baud rate generator
- Programmable clock phase and polarity
- Local loopback capability for testing
- Open-drain outputs support multiple-master configuration

19.2.2 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 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.

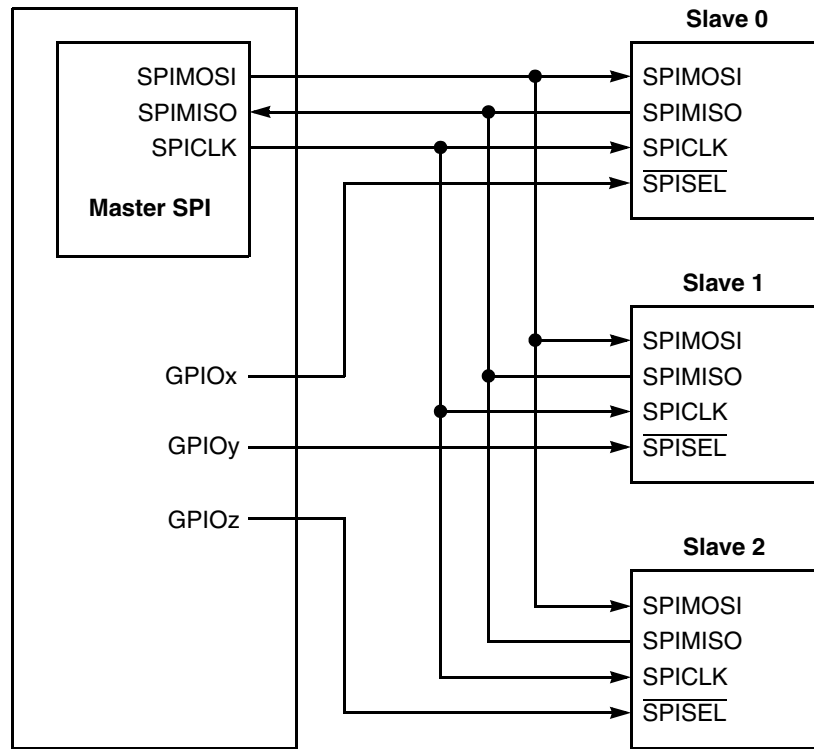


Figure 19-2. Single-Master/Multi-Slave Configuration

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 SPI as a Slave Device

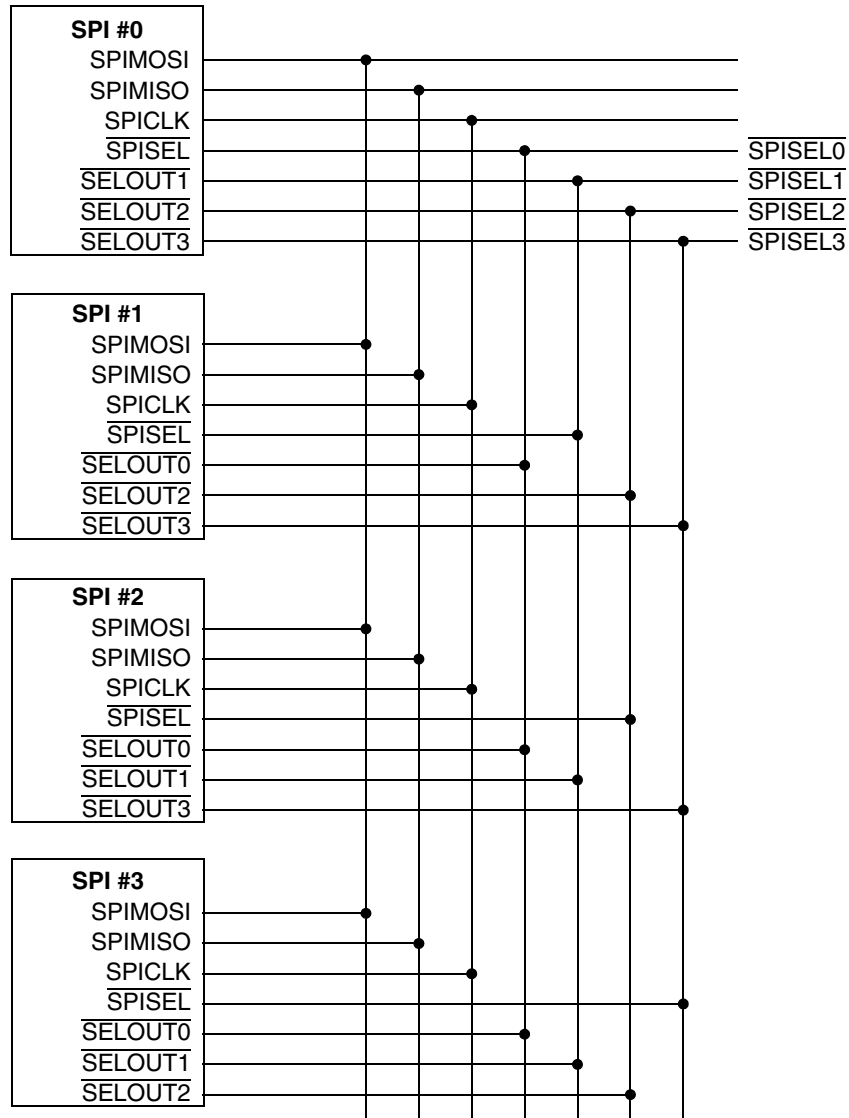
In slave mode, the SPI receives messages from an SPI master and sends a simultaneous reply. The slave's $\overline{\text{SPISEL}}$ must be asserted before Rx clocks are recognized. Once $\overline{\text{SPISEL}}$ is asserted, SPICLK becomes an input from the master to the slave. SPICLK can be any frequency from DC to input clock/2.

To prepare for data transfers, the core writes data to be sent into the SPITD register. Once $\overline{\text{SPISEL}}$ is asserted, the slave shifts data out from SPIMISO and in through SPIMOSI. The SPI sets the NF bit of the SPIE register and a maskable interrupt is issued when a full buffer finishes receiving and sending or after an error. The SPI continues reception until $\overline{\text{SPISEL}}$ is negated.

Transmission continues until no more data is available or $\overline{\text{SPISEL}}$ is negated. Transmission continues once $\overline{\text{SPISEL}}$ is reasserted and SPICLK begins toggling. After the characters in the buffer are sent, the SPI sends one as long as $\overline{\text{SPISEL}}$ remains asserted.

19.2.3.3 SPI in Multiple-Master Operation

The SPI can operate in a multiple-master environment in which all SPI devices are connected to the same bus. In this configuration, the SPIMOSI , SPIMISO , and SPICLK signals of all SPIs are shared; but the $\overline{\text{SPISEL}}$ inputs are connected separately, as shown in [Figure 19-3](#). Only one SPI device can act as master at a time—all others must be slaves. When a SPI is configured as a master, if its $\overline{\text{SPISEL}}$ input is asserted, a multiple-master error occurs because more than one SPI device is a bus master. The SPI sets $\text{SPIE}[\text{MME}]$ 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 $\text{SPMODE}[\text{EN}]$, correct the problems, and clear $\text{SPIE}[\text{MME}]$ before the SPI can be used again.



Notes:

1. All signals are open-drain.
2. For a multiple-master configuration with more than two masters, $\overline{\text{SPISEL}}$ and SPIE[MME] do not detect all possible conflicts.
3. It is the responsibility of software to arbitrate for the SPI bus (with token passing, for example).
4. SELOUTx signals are implemented in software with general-purpose I/O signals.

Figure 19-3. Multiple-Master Configuration

However, the SPI can transfer a single character at much higher rates—input clock/4 in master mode and input clock/2 in slave mode, and subjected to the timing parameters of the interconnected devices, and board trace delays. 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
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 input clock, which is generated in the device clock synthesizer.

SPICLK is a gated clock, active only during data transfers. Four combinations of SPICLK phase and polarity can be configured with the clock invert (SPMODE[CI]) and clock phase (SPMODE[CP]) register bits. SPI signals can also be configured as open-drain to support a multiple-master configuration in which a shared SPI signal is driven by the device or an external SPI device.

The SPI master-in slave-out SPIMISO signal acts as an input for master devices and as an output for slave devices. Conversely, the master-out slave-in SPIMOSI signal is an output for master devices and an input for slave devices. The dual functionality of these signals allows the SPIs in a multiple-master environment to communicate with one another using a common hardware configuration.

- When the SPI is a master, SPICLK is the clock output signal that shifts received data in from SPIMISO and transmitted data out to SPIMOSI. SPI masters must output a slave select signal to enable SPI slave devices 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.

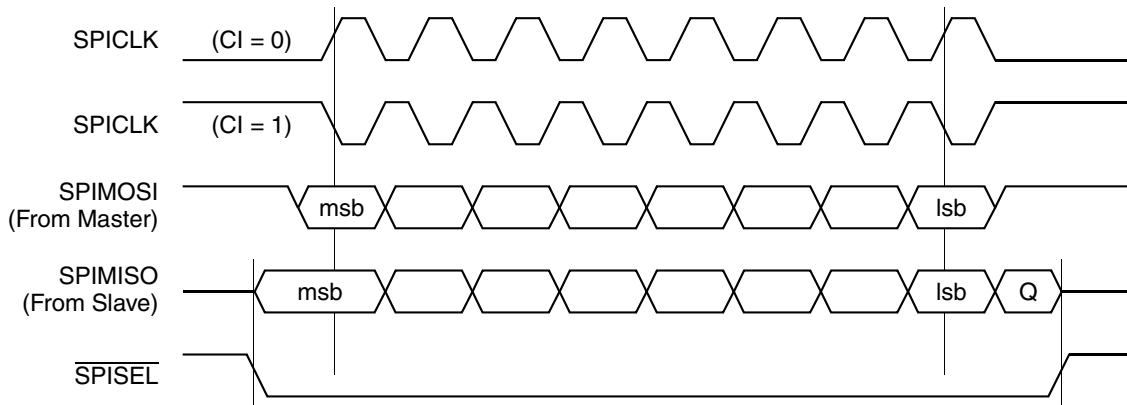
19.4 Memory Map/Register Definition

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 IMMRBAR 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 19-4. SPMODE Field Descriptions (continued)

Bits	Name	Description
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.
4	DIV16	Divide by 16. Selects the clock source for the SPI baud rate generator (SPI BRG) when configured as an SPI master. In slave mode, SPICLK is the clock source. 0 The SPI block input clock is the input to the SPI BRG. 1 The SPI block input clock/16 is the input to the SPI BRG. In slave mode this bit must be cleared.
5	REV	Reverse data mode for 8-/16-/32-bit character length only (see 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 input clocks between disabling the SPI and re-enabling. This minimal gap is sufficient provided that SPMODE[PM] and SPMODE[DIV16] are cleared during the time in which SPMODE[EN] is cleared.
8–11	LEN	Character length in bits per character. LEN can be either 32-bits, or 4- to 16-bits that are shown as follows: 0000 32-bit characters 0001–0010 Reserved, causes erratic behavior. 0011 4-bit characters ... 1111 16-bit characters The TX and RX registers (SPITD, SPIRD) hold 32 bits at a time. A character length of 32 bits fills the TX and RX registers; therefore, all of the bits in these registers are valid. However, if the character length selected by LEN is equal or less than 16 bits, then the valid bits will reside in the lower half-word of the transmit and receive registers. For example, if the character length is set to 16 bits than the valid bits will be 16–31, if the character length is set to 5 bits that the valid bits will be 16–20. Note that the transmit and receive registers each can hold only one character regardless of the character length.
12–15	PM	Prescale modulus select. Specifies the divide ratio of the prescale divider in the SPI clock generator. The SPI baud rate generator clock source (either input clock or input clock divided by 16, depending on DIV16 bit) is divided by $4 \times ([PM] + 1)$, a range from 4 to 64. The clock has a 50% duty cycle. For example, if the prescale modulus is set to PM = 0011 and DIV16 is set, the system/SPICLK clock ratio will be $16 \times (4 \times (0011 + 1)) = 256$. In slave mode this field must be cleared.
16–18	—	Reserved. Should be cleared.
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.

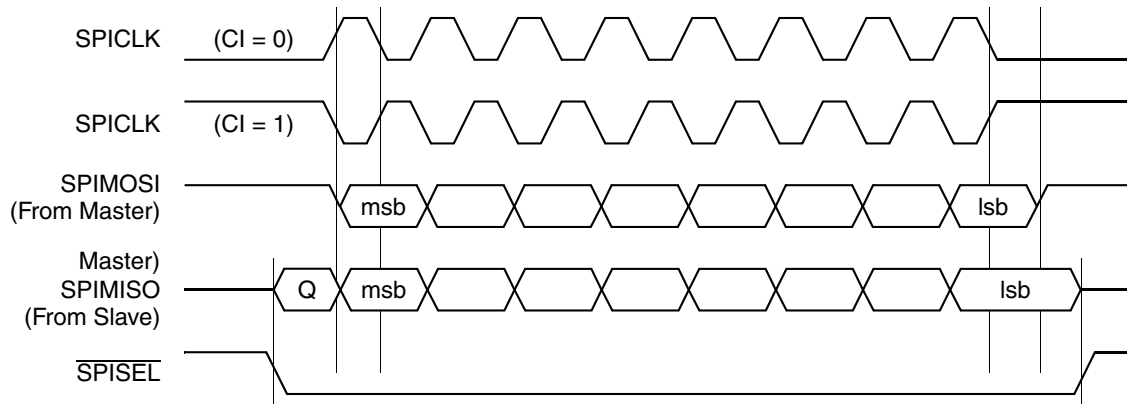
Figure 19-5 shows the SPI transfer format in which SPICLK starts toggling in the middle of the transfer (SPMODE[CP] = 0).



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

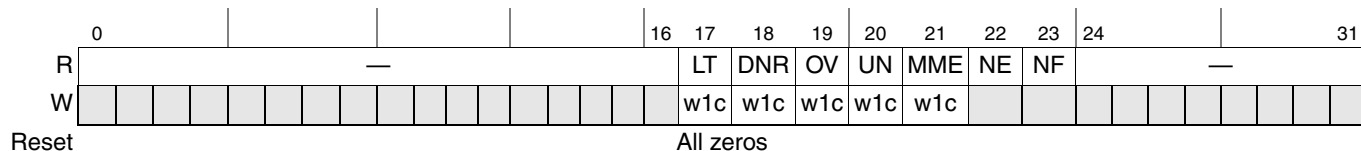


Figure 19-7. SPIE—SPI Event Register Definition

Table 19-5 describes the SPIE fields.

Table 19-5. SPIE Field Descriptions

Bits	Name	Description
0–16	—	Reserved, should be cleared.
17	LT	Last character was transmitted. The last character of the frame was completely transferred. This bit is set only if the transmitted character was the last character of the frame (if SPCOM[LST] is set). New data can be written to SPITD is indicated by bit NF.
18	DNR	Note: 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

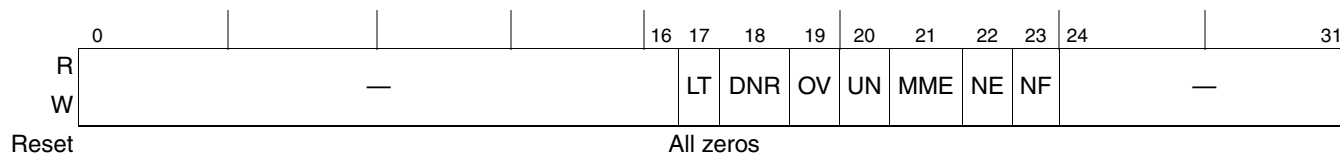


Figure 19-8. SPIM—SPI Mask Register Definition

Table 19-6 describes the SPIM fields.

Table 19-6. SPIM Field Descriptions

Bits	Name	Description
0–16	—	Reserved, should be cleared.
17	LT	Last character transmitted 0 LT event will not cause an SPI interrupt 1 LT event causes an SPI interrupt
18	DNR	In slave mode data not ready 0 Slave DNR event will not cause an SPI interrupt 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.

Offset 0x02C

Access: Write only

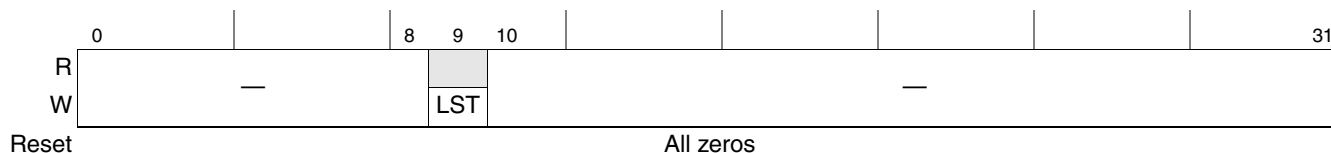


Figure 19-9. SPI Command Register Definition

[Table 19-7](#) describes the SPCOM fields.

Table 19-7. SPCOM Field Descriptions

Bits	Name	Description
0–8	—	Reserved, should be cleared.
9	LST	This bit represents the last character. Should be set before the last character is written to the SPITD. This results in SPIE[LT] being set when the character is fully transmitted and by that gives indication about the frame being fully transmitted. 0 This character is not the last character of the frame 1 This character is the last character of the frame
10–31	—	Reserved, should be cleared.

19.4.1.5 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.

Offset 0x030

Access: Write only

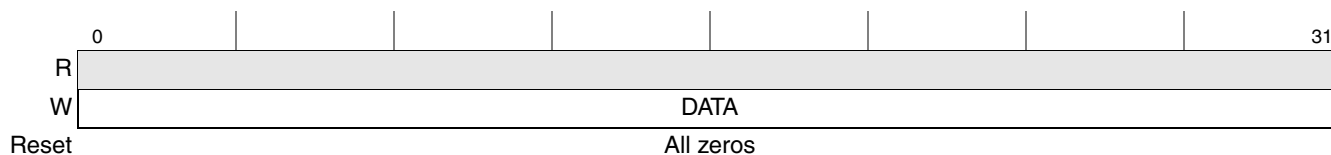


Figure 19-10. SPI Transmit Data Hold Register Definition

[Table 19-8](#) shows the field descriptions of the SPI transmit data hold register.

Table 19-8. SPI Transmit Data Hold Field Descriptions

Bits	Name	Description
0–31	DATA	These bits are the data to be sent.

19.4.1.6 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.

Offset 0x034

Access: Read-only

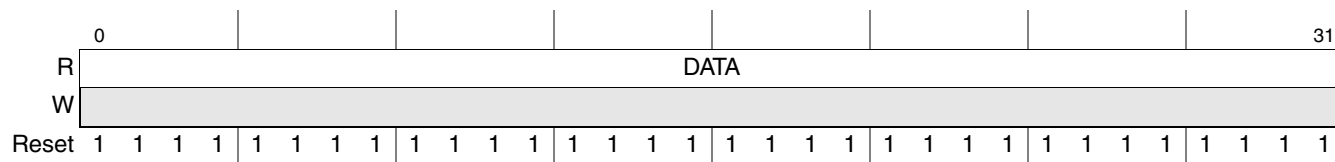


Figure 19-11. SPI Receive Data Hold Register Definition

[Table 19-9](#) shows the field descriptions of the SPI receive data hold register.

Table 19-9. SPI Receive Data Hold Field Descriptions

Bits	Name	Description
0–31	DATA	Received data. These bits are the received data from the SPI bus.

19.4.1.6.1 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 bits (SPMODE[LEN] = 7).

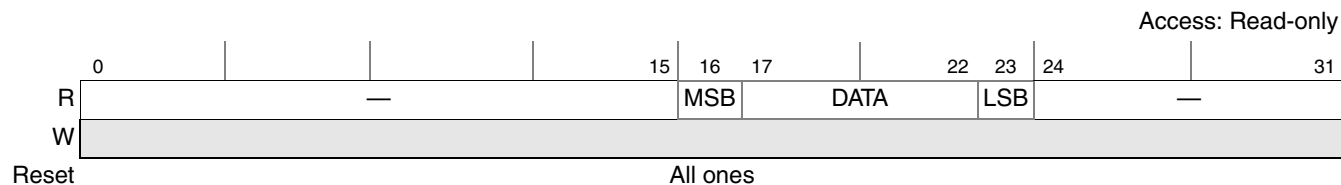


Figure 19-12. Example SPMODE[REV] = 0 SPMODE[LEN] = 7 LSB Sent First

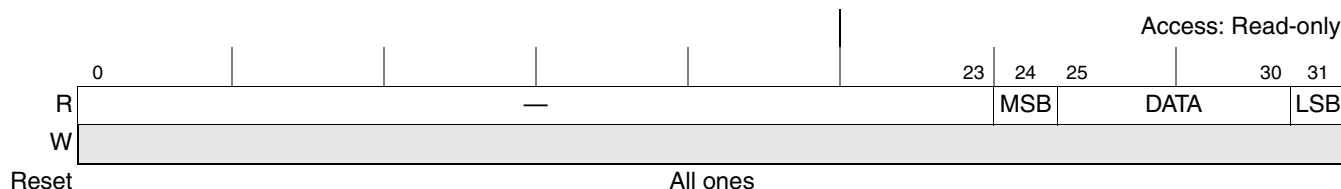


Figure 19-13. Example SPMODE[REV] = 1 SPMODE[LEN] = 7 MSB Sent First

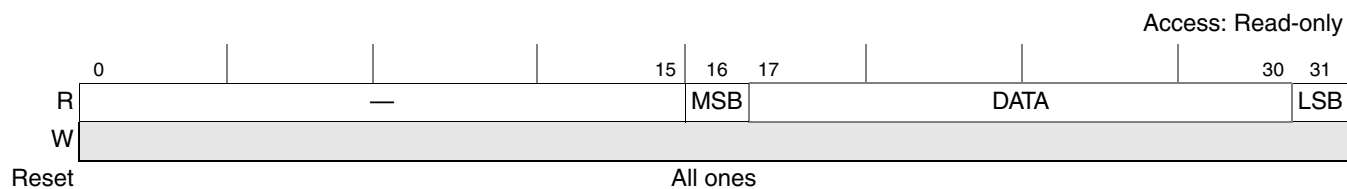


Figure 19-14. Example SPMODE[REV] = 1 SPMODE[LEN] = 15 MSB Sent First

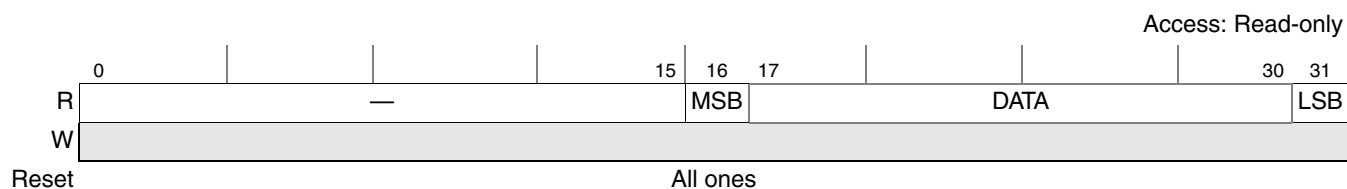


Figure 19-15. Example SPMODE[REV] = 0 SPMODE[LEN] = 15 LSB Sent First

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 0xFFFF_FFFF to SPIE to clear any previous events. Configure SPIM to enable all desired SPI interrupts.
3. Configure SPMODE to enable normal operation (not loopback), master mode, SPI enabled, character length, and the fastest speed possible.
4. Write the first character to be sent to SPITD.

19.5.2 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 0xFFFF_FFFF to SPIE to clear any previous events.
2. Configure SPIM to enable all desired SPI interrupts.
3. Configure SPMODE to enable normal operation (not loopback), slave mode, SPI enabled, and characters length.
4. Write the first data to be sent to SPITD, to enable the SPI to be ready once the master begins to transfer.

Chapter 20

JTAG/Testing Support

20.1 Overview

The device provides a JTAG (Joint Test Action Group) interface to facilitate boundary-scan testing. The JTAG interface complies to the IEEE 1149.1 boundary-scan specification. For additional information about JTAG operations, refer to the IEEE 1149.1 specification.

The JTAG interface consists of a set of five signals, three JTAG registers (see [Section 20.3, “JTAG Registers and Scan Chains,”](#)) and a test access port (TAP) controller, described in the following sections. A block diagram of the JTAG interface is shown in [Figure 20-1](#).

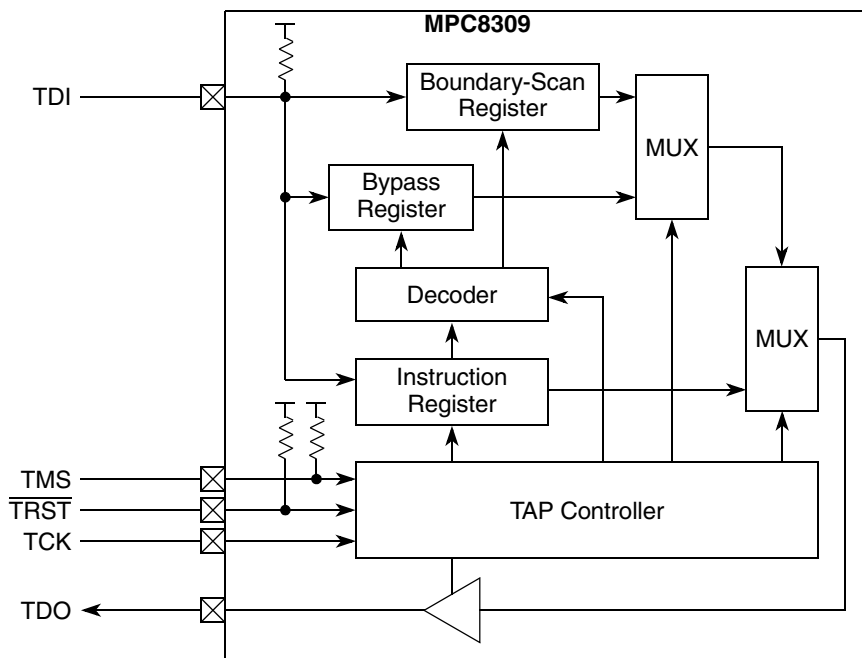


Figure 20-1. JTAG Interface Block Diagram

20.2 JTAG Signals

The device provides the following five dedicated JTAG signals:

- Test data input (TDI)
- Test data output (TDO)
- Test mode select (TMS)

- Test reset ($\overline{\text{TRST}}$)
- Test clock (TCK)

The TDI and TDO signals input and output all instructions and data to the JTAG scan registers. JTAG operations are controlled by the TAP controller through the TMS and TCK signals. Boundary-scan data is latched by the TAP controller on the rising edge of the TCK signal. The $\overline{\text{TRST}}$ signal is specified as optional by the IEEE 1149.1 specification, and is used to reset the TAP controller asynchronously. The assertion of the $\overline{\text{TRST}}$ signal at power-on reset ensures that the JTAG logic does not interfere with the normal operation of the device.

20.2.1 External Signal Descriptions

The JTAG signals are summarized in [Table 20-1](#).

Table 20-1. JTAG Test Signals Summary

Name	Description	Functional Block	Function	Reset Value	I/O
TCK	Test clock	Debug	Clock for JTAG testing.	—	I
TDI	Test data input		Serial input for instructions and data to the JTAG test subsystem. Internally pulled up.	—	I
TDO	Test data output		Serial data output for the JTAG test subsystem. High impedance except when scanning out data.	High impedance	O
TMS	Test mode select		Carries commands to the TAP controller for boundary scan operations. Internally pulled up.	—	I
$\overline{\text{TRST}}$	Test reset		Resets the TAP controller asynchronously. Internally pulled up.	—	I

[Table 20-2](#) shows detailed descriptions of the JTAG test signals.

Table 20-2. JTAG Test—Detailed Signal Descriptions

Signal	I/O	Description	
TCK	I	JTAG test clock.	
		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.
		Timing	See IEEE 1149.1 specification 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 specification for more details.

Table 20-2. JTAG Test—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
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 specification for more details.
TMS	I	JTAG test mode select.	
		State Meaning	Asserted/Negated—Decoded by the internal JTAG TAP controller to distinguish the primary operation of the test support circuitry. An unterminated input appears as a high signal level to the test logic due to an internal pull-up resistor.
		Timing	See IEEE 1149.1 specification for more details.
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 specification for more details.

20.3 JTAG Registers and Scan Chains

The bypass, boundary-scan, and instruction JTAG registers and their associated scan chains are mandatory for 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 device. The use of the bypass register reduces the total scan string size of the boundary-scan test.
- Boundary-scan registers. The JTAG interface provides a chain of registers dedicated to boundary-scan operations. To be JTAG-compliant, these registers cannot be shared with any functional registers of the device. The boundary-scan register chain includes registers controlling the direction of the input/output drivers, in addition to the registers reflecting the signal value received or driven.

The boundary-scan registers capture the input or output state of the device's signals during a Capture_DR TAP controller state. When a data scan is initiated following the Capture_DR state, the sampled values are shifted out through the TDO output while new boundary-scan register values are shifted in through the TDI input. At the end of the data scan operation, the boundary-scan registers are updated with the new values during an update_DR TAP controller state.

- Instruction register. The 8-bit JTAG instruction register serves as an instruction and status register. As TAP controller instructions are scanned in through the TDI input, the TAP controller status bits are scanned out through the TDO output.

- TAP controller. The device provides a standard JTAG TAP controller that controls instruction and data scan operations. The TMS signal controls the state transitions of the TAP controller.

Chapter 21

General Purpose I/O (GPIO)

21.1 Introduction

This chapter describes the general-purpose I/O module, including pin descriptions, register settings, and interrupt capabilities. The device includes two GPIO modules. Figure 21-1 shows the block diagram of one GPIO module.

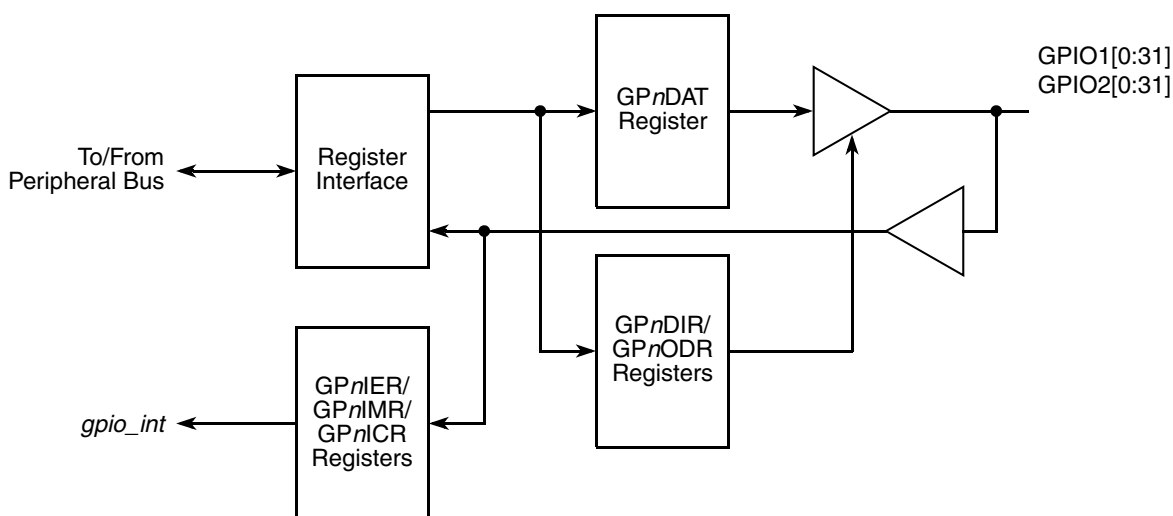


Figure 21-1. GPIO_n Module Block Diagram

NOTE

GPIO1[0:31] represents the signals GPIO[0] to GPIO[31], and GPIO2[0:31] represents signals GPIO[32] to GPIO[63] throughout this chapter.

21.1.1 Overview

Each GPIO module supports 32 general-purpose I/O ports. The MPC8309 has two GPIO modules for a total of 64 GPIO ports. Each port can be configured as an input or as an output. If a port is configured as an input, it can optionally generate an interrupt on detection of a change. If a port is configured as an output, it can be individually configured as an open-drain or a fully active output.

21.1.2 Features

The GPIO unit implements the following features:

- 64 input/output ports
- All signals, including the package pins that are muxed with other functions, are configured as inputs when the device comes out of reset and also when $\overline{\text{HRESET}}$ is asserted. The pins GPIO[0:6], GPIO[16:22], and GPIO[32:39] use multisite muxing. This is further described in the Signal Description chapter.
- Open-drain capability on all ports
- All ports can optionally generate an interrupt upon changing their state.

21.2 External Signal Description

The following section provides information about GPIO signals.

21.2.1 Signals Overview

Table 21-1 provides detailed descriptions of the external GPIO signals.

Table 21-1. IPIC 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 address held in IMMRBAR as defined in Chapter 2, “Memory Map.”

Table 21-2 shows the memory map of GPIO.

Table 21-2. GPIO Register Address Map

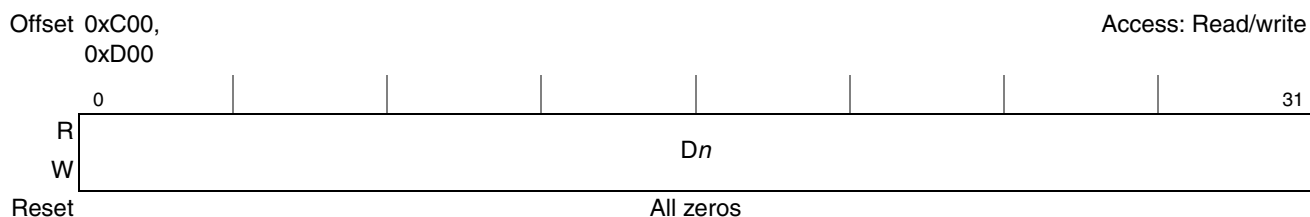
Offset	Register	Access	Reset Value	Section/Page
General Purpose I/O (GPIO1)—Block Base Address 0x0_0C00 General Purpose I/O (GPIO2)—Block Base Address 0x0_0D00				
0xC00	GPIO1 direction register (GP1DIR)	R/W	0x0000_0000	21.3.1/21-3
0xC04	GPIO1 open drain register (GP1ODR)	R/W	0x0000_0000	21.3.2/21-4

Table 21-2. GPIO Register Address Map (continued)

Offset	Register	Access	Reset Value	Section/Page
0xC08	GPIO1 data register (GP1DAT)	R/W	0x0000_0000	21.3.3/21-4
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)	w1c	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_nDIR)

[Table 21-3](#) defines the bit fields of GP_nDIR.

Table 21-3. GP_nDIR Bit Settings

Bits	Name	Description
0–31	<i>D_n</i>	Direction. Indicates whether a signal is used as an input or an output. 0 The corresponding signal is an input. 1 The corresponding signal is an output.

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.

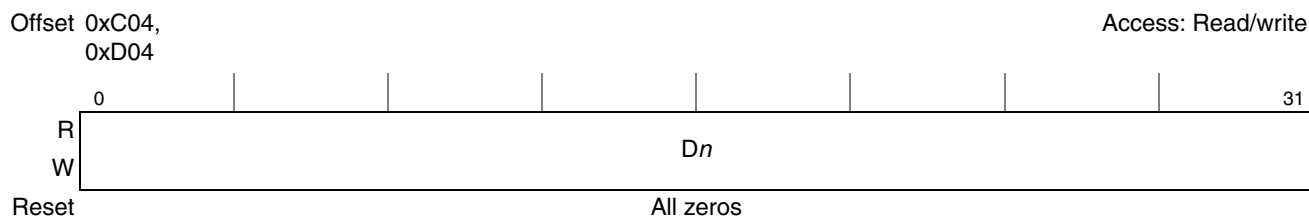


Figure 21-3. GPIO_n Open Drain Register (GP1ODR–GP2ODR)

Table 21-4 defines the bit fields of GP_nODR.

Table 21-4. GP_nODR Bit Settings

Bits	Name	Description
0–31	<i>D_n</i>	Open-drain configuration. Indicates whether a signal is actively driven as an output or an open-drain driver. This register has no effect on signals programmed as inputs in the corresponding GP _n DIR. 0 The I/O signal is actively driven as an output. Note: The I/O signal is an open-drain driver. As an output, the signal is driven active-low, otherwise it is three-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.

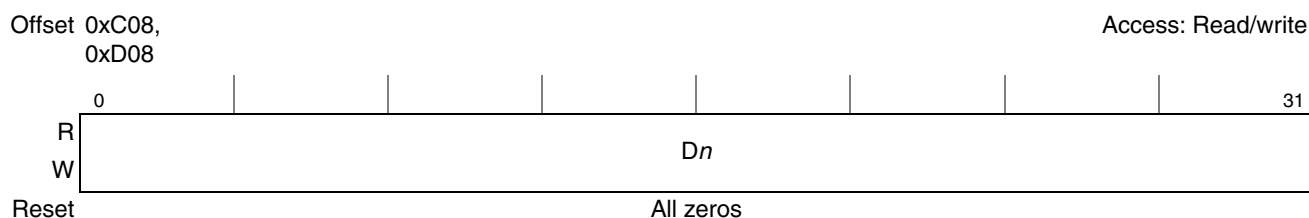


Figure 21-4. GPIO_n Data Register (GP1DAT–GP2DAT)

Table 21-5 defines the bit fields of GP_nDAT.

Table 21-5. GP_nDAT Bit Settings

Bits	Name	Description
0–31	<i>D_n</i>	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.

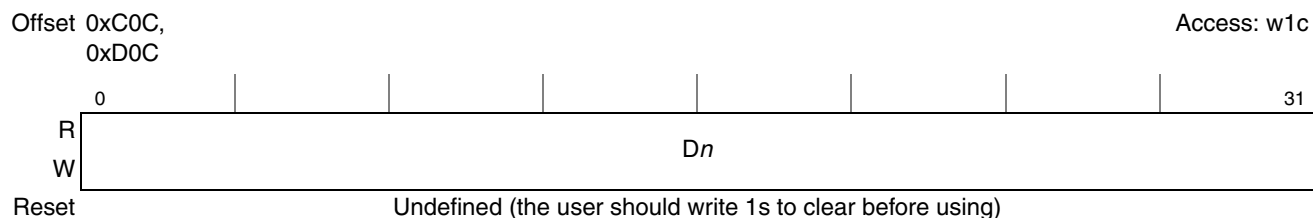


Figure 21-5. GPIO n Interrupt Event Register (GP1IER–GP2IER)

[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 an interrupt event occurred on the corresponding GPIO signal. 0 No interrupt event occurred on the corresponding GPIO signal. 1 Interrupt event occurred on the corresponding GPIO signal.

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.

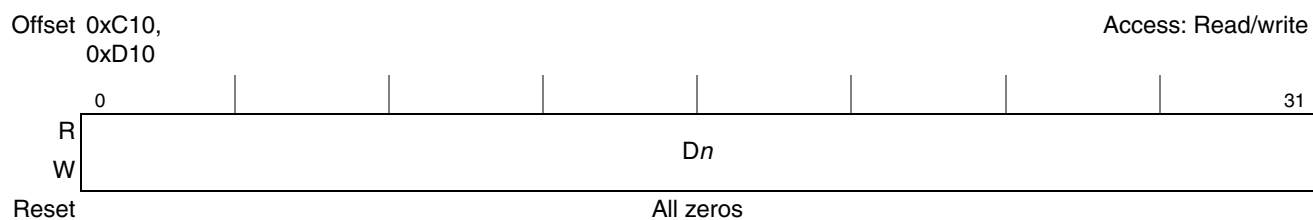


Figure 21-6. GPIO n Interrupt Mask Register (GP1IMR–GP2IMR)

[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 on either a high-to-low change or any change on the state of the signal.

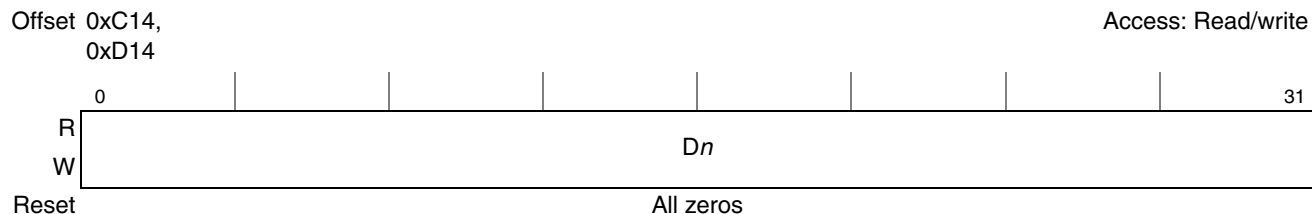


Figure 21-7. GPIO n Interrupt Control Register (GP1ICR–GP2ICR)

[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 Sequencer

22.1 Sequencer Overview

The I/O sequencer switches transactions among its ports, using a buffer pool to minimize blocking. [Figure 22-1](#) is a block diagram of the I/O sequencer (IOS).

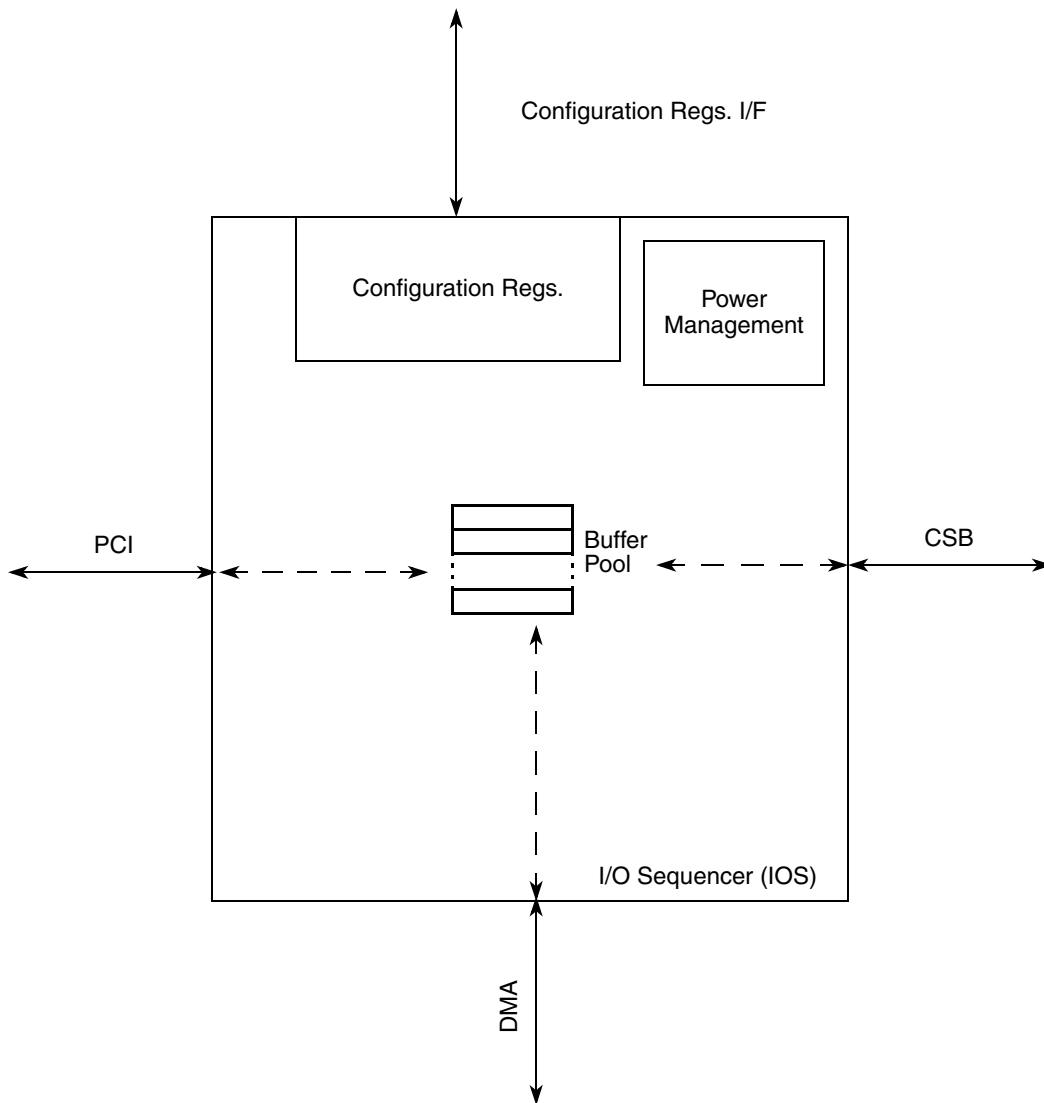


Figure 22-1. I/O Sequencer Block Diagram

22.1.1 Sequencer Features

The I/O sequencer includes the following features:

- Switches transactions among its ports
- Contains 8 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.

22.2 Sequencer External Signal Description

The I/O sequencer has no external signals.

22.3 Sequencer Memory Map/Register Definition

Table 22-1 shows the I/O sequencer memory map.

Table 22-1. Sequencer Memory Map

Offset	Register	Access	Reset	Section/Page
I/O Sequencer (IOS)—Block Base Address 0x0_8400				
0x00	POTAR0—PCI outbound translation address register 0	R/W	All zeros	22.4.1/22-3
0x08	POBAR0—PCI outbound base address register 0	R/W	All zeros	22.4.2/22-3
0x10	POCMR0—PCI outbound comparison mask register 0	R/W	All zeros	22.4.3/22-4
0x18	POTAR1—PCI outbound translation address register 1	R/W	All zeros	22.4.1/22-3
0x20	POBAR1—PCI outbound base address register 1	R/W	All zeros	22.4.2/22-3
0x28	POCMR1—PCI outbound comparison mask register 1	R/W	All zeros	22.4.3/22-4
0x30	POTAR2—PCI outbound translation address register 2	R/W	All zeros	22.4.1/22-3
0x38	POBAR2—PCI outbound base address register 2	R/W	All zeros	22.4.2/22-3
0x40	POCMR2—PCI outbound comparison mask register 2	R/W	All zeros	22.4.3/22-4
0x48	POTAR3—PCI outbound translation address register 3	R/W	All zeros	22.4.1/22-3
0x50	POBAR3—PCI outbound base address register 3	R/W	All zeros	22.4.2/22-3
0x58	POCMR3—PCI outbound comparison mask register 3	R/W	All zeros	22.4.3/22-4
0x60	POTAR4—PCI outbound translation address register 4	R/W	All zeros	22.4.1/22-3
0x68	POBAR4—PCI outbound base address register 4	R/W	All zeros	22.4.2/22-3
0x70	POCMR4—PCI outbound comparison mask register 4	R/W	All zeros	22.4.3/22-4
0x78	POTAR5—PCI outbound translation address register 5	R/W	All zeros	22.4.1/22-3
0x80	POBAR5—PCI outbound base address register 5	R/W	All zeros	22.4.2/22-3

Table 22-1. Sequencer Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x88	POCMR5—PCI outbound comparison mask register 5	R/W	All zeros	22.4.3/22-4
0xF0	PMCR—Power management control register	R/W	0x0000_0001	22.4.4/22-5
0xF8	DTCR—Discard timer control register	R/W	All zeros	22.4.5/22-6

22.4 Sequencer Register Descriptions

22.4.1 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 22-2](#) shows the POTAR_n register fields.



Figure 22-2. PCI Outbound Translation Address Registers (POTAR_n)

[Table 22-2](#) describes POTAR_n fields.

Table 22-2. 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. The translation address must be aligned based on the window's size.

22.4.2 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 22-3](#) shows the POBAR_n register fields.



Figure 22-3. PCI Outbound Base Address Registers (POBAR_n)

Table 22-3 describes POBAR_n fields.

Table 22-3. 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.

22.4.3 PCI Outbound Comparison Mask Registers (POCMR_n)

The PCI outbound comparison mask register (POCMR_n) 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 22.5.1, “Transaction Forwarding,” for more information. Figure 22-4 shows the POCMR_n register fields.

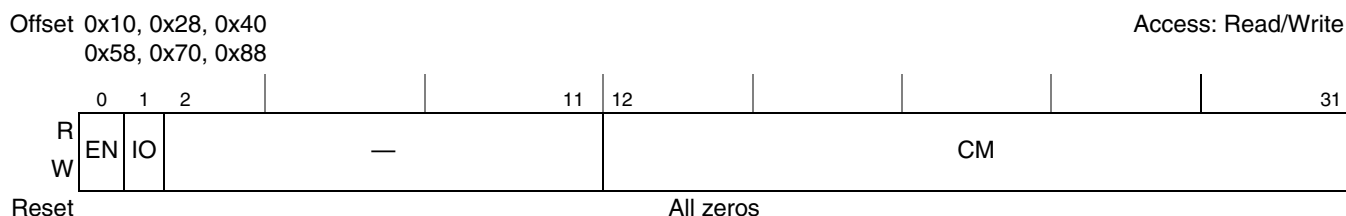


Figure 22-4. PCI Outbound Comparison Mask Registers (POCMR_n)

Table 22-4 describes the bit settings of the POCMR_n register.

Table 22-4. 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
2–11	—	Reserved, should be cleared.

22.4.5 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 22-6 shows the DTCR register fields.

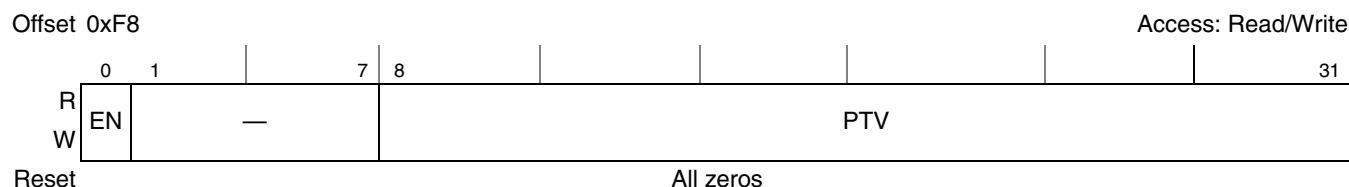


Figure 22-6. Discard Timer Control Register (DTCR)

Table 22-6 describes DTCR fields.

Table 22-6. 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 $(2^{24} - \text{PTV})$ internal clock cycles if the master has not repeated the transaction. 0xFFFFFFFF 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, <ul style="list-style-type: none"> Assuming the internal frequency is twice the PCI frequency The PTV should equal $2^{24} - 2^{16}$ (0xFF0000).

22.5 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 8 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 in order.

22.5.1 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.

22.5.1.1 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 the PCI controller, the transaction is forwarded to the PCI port. These address values are configuration options of the I/O sequencer. See [Table 23-3](#) 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 the PCI port, with the address translated. See [Section 22.5.2, “PCI Outbound Address Translation,”](#) for more information.

22.5.1.2 Transactions from the PCI Port

Transactions from the PCI port are forwarded as follows:

- 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.

22.5.1.3 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 the PCI port, with the address translated. See [Section 22.5.2, “PCI Outbound Address Translation,”](#) for more information.
- All other transactions are forwarded to the CSB port.

22.5.2 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 22.4.2, “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 22-7 shows an example translation window for outbound memory accesses.

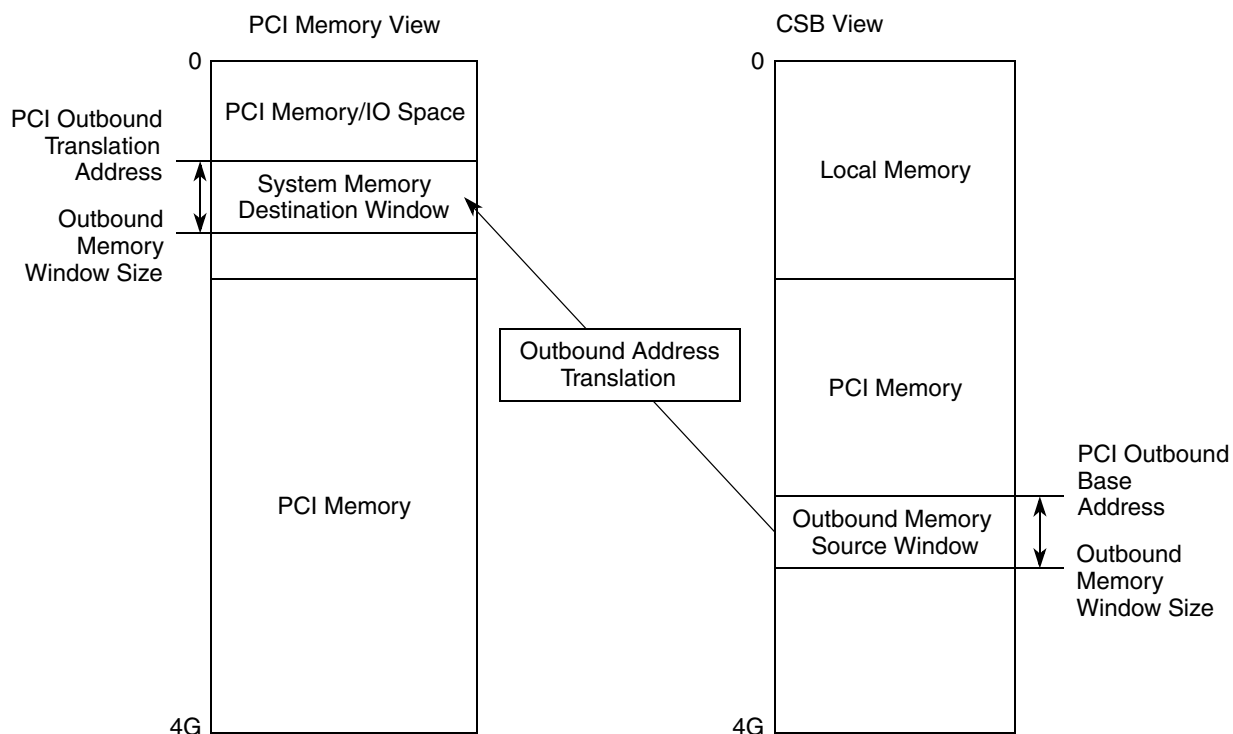


Figure 22-7. Outbound PCI Memory Address Translation

The six sets of outbound translation registers allow six simultaneous translation windows to the PCI port. 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.

22.5.3 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 the 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 the PCI port without forcing the PCI port to first accept a read.

Chapter 23

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.

23.1 PCI 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.

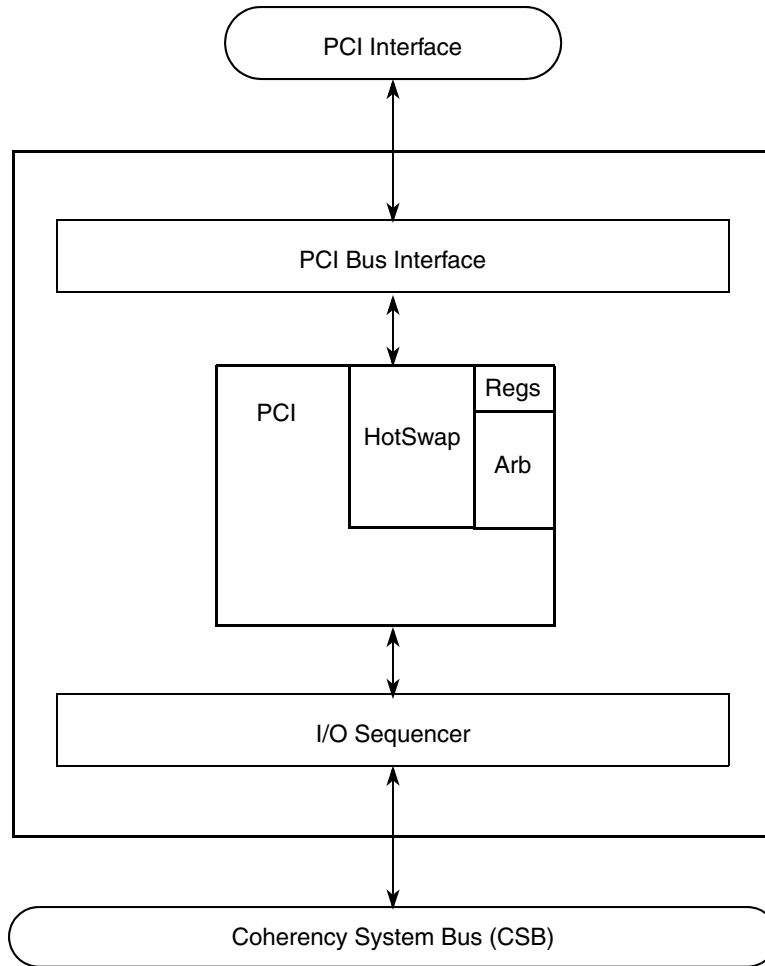


Figure 23-1. PCI Controller Block Diagram

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-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 PCI supports up to three external masters.

The PCI 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 23.4.4.4, “Host Mode Configuration Access,”](#) for more information. 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.

The PCI interface does not flush pending outbound writes as a result of an inbound read command. Systems must not rely on inbound reads to ensure all pending outbound writes have completed. For example, consider the case where a core writes data to a PCI device and then updates a flag in the local

DDR memory indicating the write to PCI has completed. An external PCI master may misread the flag ahead of the actual write transaction's completion on the PCI bus.

23.1.1 PCI Features

The PCI controller includes the following features:

- PCI specification revision 2.3 compliant
- 32-bit PCI interface support
- Host and agent mode support
- 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 three masters on PCI
- 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

23.1.2 PCI Modes of Operation

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 23-1](#) summarizes these modes.

Table 23-1. PCI Controller Modes

Parameter	Description	Section/Page
Host/agent configuration	Selects between host and agent mode for the PCI interface.	4.3.2.2/4-14
PCI arbiter enable	Enables the on-chip PCI bus arbiter	4.3.2.2/4-14

23.1.2.1 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, “Boot Memory Space \(BMS\).”](#)

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 23.3.3.24, “PCI Function Configuration Register”](#)), either from reset configuration or after being cleared by software.

23.1.2.2 PCI Arbiter Configuration

The interface can be configured to use an on-chip or off-chip PCI arbiter. Arbitration for PCI is determined by the value in RCWH[PCIARB]. See [Section 4.3.2.2, “Reset Configuration Word High Register \(RCWHR\)”](#) for more information.

23.2 PCI External Signal Description

Table 13-2 shows the properties of the PCI signals.

Table 23-2. 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	—
PCI_AD[31:0]	PCI address / data	High impedance	—
PCI_C/BE[3:0]	PCI bus command / byte enable	High impedance	—
PCI_DEVSEL	PCI device select	High impedance	Required
PCI_FRAME	PCI cycle frame	High impedance	Required
PCI_REQ[0:2]	PCI arbiter requests	Configuration-dependent	Required on inputs
PCI_GNT[0:2]	PCI arbiter grants	Configuration-dependent	—
PCI_IDSEL	PCI initialization device select	—	—
PCI_INTA	PCI interrupt A	High impedance	Required
PCI_IRDY	PCI initiator ready	High impedance	Required
PCI_PAR	PCI parity	High impedance	—
PCI_PERR	PCI parity error	High impedance	Required
PCI_RESET_OUT	PCI reset output	Asserted	
PCI_SERR	PCI system error	High impedance	Required
PCI_STOP	PCI stop	High impedance	Required
PCI_TRDY	PCI target ready	High impedance	Required

Figure 23-2 shows the external PCI signals.

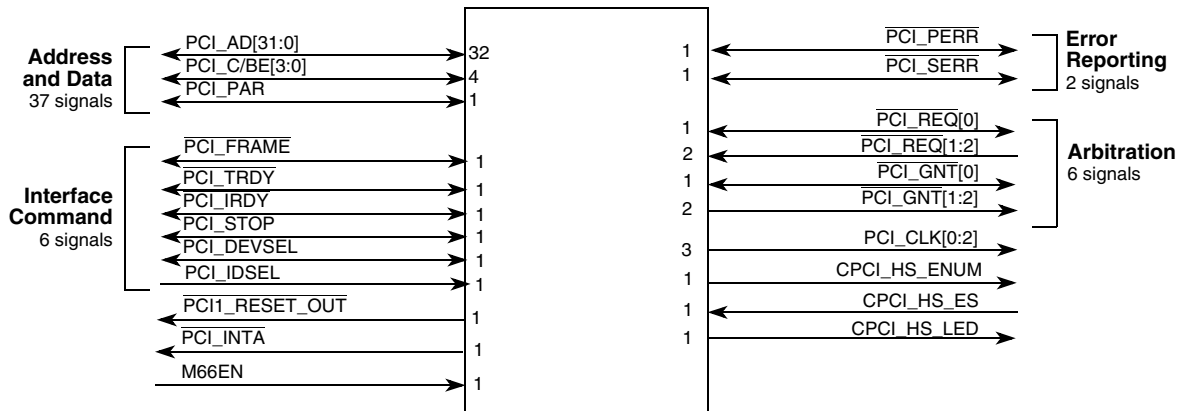


Figure 23-2. PCI Interface External Signals

Table 23-3 contains detailed descriptions of the external PCI interface signals.

Table 23-3. PCI 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
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 Asserted—The PCI interface signals use the 66-MHz PCI AC timing parameters. Negated—The PCI interface signals use the 33-MHz PCI AC timing parameters.
		Timing Assertion/Negation—Constant

Table 23-3. PCI Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
PCI_AD[31:0]	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.	
	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 and, PCI_AD[31:24] define the MSB.
		Timing	Assertion/Negation—As specified by <i>PCI Local Bus Specification Rev 2.3</i>
	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 and, PCI_AD[31:24] define the MSB.
Timing		Assertion/Negation—As specified by <i>PCI Local Bus Specification Rev 2.3</i>	
PCI_C/BE[3:0]	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[3:0], 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.
		Timing	Assertion/Negation—As specified by <i>PCI Local Bus Specification Rev 2.3</i>
	I	Inputs for the bi-directional command/byte enable.	
		State Meaning	Asserted/Negated—During the address phase, PCI_CBE[3:0], indicate the command that another master is sending. During the PCI bus data phase, PCI_CBE[3:0], indicate which byte lanes are valid.
Timing		Assertion/Negation—As specified by <i>PCI Local Bus Specification Rev 2.3</i>	
PCI_DEVSEL	I/O	PCI device select.	
	O	Outputs for the bi-directional device select.	
		State Meaning	Asserted—The PCI controller has decoded the address and is the target of the current access. Negated—The PCI controller has decoded the address and is not the target of the current access.
		Timing	Assertion/Negation—As specified by <i>PCI Local Bus Specification Rev 2.3</i>
	I	Inputs for the bi-directional device select.	
		State Meaning	Asserted—Some PCI agents (other than this PCI 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 <i>PCI Local Bus Specification Rev 2.3</i>	

Table 23-3. PCI Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
PCI_FRAME	I/O	PCI 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 controller acting as a PCI master which is initiating a bus transaction. While PCI_FRAME is asserted, data transfers may continue. Negated—If PCI_IRDY is asserted, indicates that the PCI transaction is in the final data phase; if PCI_IRDY is negated, indicates that the PCI bus is idle.
		Timing	Assertion/Negation—As specified by <i>PCI Local Bus Specification Rev 2.3</i>
	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 <i>PCI Local Bus Specification Rev 2.3</i>	
PCI_GNT0	I/O	PCI arbiter grants. Output signal on this PCI 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 controller granted control of the PCI bus to agent 0. Negated—The PCI controller did not grant control of the PCI bus to agent 0.
		Timing	Assertion/Negation—As specified by <i>PCI Local Bus Specification Rev 2.3</i>
	I	Inputs for the bi-directional arbiter grants.	
		State Meaning	Asserted—The PCI controller has been granted control of the PCI bus by an external arbiter. Negated—The PCI controller has not been granted control of the PCI bus by an external arbiter.
Timing		Assertion/Negation—As specified by <i>PCI Local Bus Specification Rev 2.3</i>	
PCI_GNT[1:2]	O	PCI arbiter grants. Output signals on this PCI 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 controller granted control of the PCI bus to agent n . Negated—The PCI controller did not grant control of the PCI bus to agent n .
		Timing	Assertion/Negation—As specified by <i>PCI Local Bus Specification Rev 2.3</i>
PCI_IDSEL	I	PCI 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 PCI controller is being selected as a target of a configuration read or write transactions. Negated—The PCI controller is not being selected as a target of configuration read or write transactions.
		Timing	Assertion/Negation—As specified by <i>PCI Local Bus Specification Rev 2.3</i>

Table 23-3. PCI Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description
$\overline{\text{PCI_INTA}}$	O	PCI interrupt A. It can be disabled from PCI Command Configuration Register. For more information, see Section 23.3.3.3, “PCI Command Configuration Register” and Table 14-6, ODR Field Descriptions .
	State Meaning	Asserted—The PCI controller signals an interrupt to the PCI host. Negated—The PCI controller is not currently signalling an interrupt.
	Timing	Assertion/Negation—As specified by <i>PCI Local Bus Specification Rev 2.3</i>
$\overline{\text{PCI_IRDY}}$	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 controller, acting as a PCI master, can complete the current data phase of a PCI transaction. During a write, this PCI controller asserts $\overline{\text{PCI_IRDY}}$ to indicate that valid data is present on $\text{PCI_AD}[31:0]$. During a read, this PCI controller asserts $\overline{\text{PCI_IRDY}}$ to indicate that it is prepared to accept data. Negated—The PCI target needs to wait before this PCI controller, acting as a PCI master, can complete the current data phase. During a write, this PCI controller negates $\overline{\text{PCI_IRDY}}$ to insert a wait cycle when it cannot provide valid data to the target. During a read, this PCI controller negates $\overline{\text{PCI_IRDY}}$ to insert a wait cycle when it cannot accept data from the target.
	Timing	Assertion/Negation—As specified by <i>PCI Local Bus Specification Rev 2.3</i>
	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 $\overline{\text{PCI_FRAME}}$ is asserted, indicates a wait cycle from another master. If $\overline{\text{PCI_FRAME}}$ is negated, indicates that the PCI bus is idle.
PCI_PAR	I/O	PCI parity.
	O	Outputs for the bi-directional parity.
	State Meaning	Asserted—Odd parity across $\text{PCI_AD}[31:0]$ and $\text{PCI_CBE}[3:0]$ during address and data phases. Negated—Even parity across $\text{PCI_AD}[31:0]$ and $\text{PCI_AD}[31:0]$ during address and data phases.
	Timing	Assertion/Negation—As specified by <i>PCI Local Bus Specification Rev 2.3</i>
	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 <i>PCI Local Bus Specification Rev 2.3</i>	

Table 23-3. PCI Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
$\overline{\text{PCI_PERR}}$	I/O	PCI parity error	
	O	Outputs for the bi-directional parity error.	
		State Meaning	Asserted—The PCI controller 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 <i>PCI Local Bus Specification Rev 2.3</i>
	I	Inputs for the bi-directional parity error.	
		State Meaning	Asserted—Another PCI agent detects a data parity error while this PCI controller is sourcing data (this PCI 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 <i>PCI Local Bus Specification Rev 2.3</i>	
$\overline{\text{PCI_REQ0}}$	I/O	PCI bus request. Input signal on this PCI 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 <i>PCI Local Bus Specification Rev 2.3</i>
	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 <i>PCI Local Bus Specification Rev 2.3</i>	
$\overline{\text{PCI_REQ}}[1:2]$	I	PCI bus request. Input signals on this PCI controller when the arbiter is enabled. Note that $\overline{\text{PCI_REQ}}[n]$ is a point-to-point signal. Every master has its own bus request signal. Following is the state meaning for the $\overline{\text{PCI_REQ}}[n]$ input.	
		State Meaning	Asserted—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 <i>PCI Local Bus Specification Rev 2.3</i>
$\overline{\text{PCI_RESET_OUT}}$	O	PCI 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 <i>PCI Local Bus Specification Rev 2.3</i>

Table 23-3. PCI Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
$\overline{\text{PCI_SERR}}$	I/O	PCI system error	
	O	Outputs for the bi-directional system error.	
		State Meaning	Asserted—An address parity error, a target-abort (when this PCI controller is acting as the initiator), or some other system error (where the result is a catastrophic error) was detected. Negated—No error.
		Timing	Assertion/Negation—As specified by <i>PCI Local Bus Specification Rev 2.3</i>
	I	Inputs for the bi-directional system error.	
		State Meaning	Asserted—A device (other than this PCI controller) has detected a catastrophic error. Negated—No error.
Timing		Assertion/Negation—As specified by <i>PCI Local Bus Specification Rev 2.3</i>	
$\overline{\text{PCI_STOP}}$	I/O	PCI stop.	
	O	Outputs for the bi-directional stop.	
		State Meaning	Asserted—The PCI 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 <i>PCI Local Bus Specification Rev 2.3</i>
	I	Inputs for the bi-directional stop.	
		State Meaning	Asserted—A target is requesting that this PCI controller, when acting as initiator, stop the current transaction. Negated—The current transaction can continue. Note: $\overline{\text{PCI_STOP}}$ is a bus signal, and it can be asserted and as an input to this PCI controller while this PCI controller has nothing to do w/ the transaction at all.
Timing		Assertion/Negation—As specified by <i>PCI Local Bus Specification Rev 2.3</i>	
$\overline{\text{PCI_TRDY}}$	I/O	PCI target ready.	
	O	Outputs for the bi-directional target ready.	
		State Meaning	Asserted—The PCI controller, acting as a PCI target, can complete the current data phase of a PCI transaction. During a read, this PCI controller asserts $\overline{\text{PCI_TRDY}}$ to indicate that valid data is present on $\text{PCI_AD}[31:0]$. During a write, this PCI controller asserts $\overline{\text{PCI_TRDY}}$ to indicate that it is prepared to accept data. Negated—The PCI initiator needs to wait before this PCI controller, acting as a PCI target, can complete the current data phase. During a read, this PCI controller negates $\overline{\text{PCI_TRDY}}$ to insert a wait cycle when it cannot provide valid data to the initiator. During a write, this PCI controller negates $\overline{\text{PCI_TRDY}}$ to insert a wait cycle when it cannot accept data from the initiator.
		Timing	Assertion/Negation—As specified by <i>PCI Local Bus Specification Rev 2.3</i>
	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 <i>PCI Local Bus Specification Rev 2.3</i>	

23.3 PCI 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 23-4](#), 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 23-5](#). They can be accessed by PCI masters via the PCI controller to the CSB through the PIMMR inbound window. Note that [Table 23-5](#) does not list outbound address translation registers; these are contained in the I/O sequencer (IOS) memory-mapped registers. See [Chapter 22, “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 23.3.3, “PCI Configuration Space Registers.”](#)

Table 23-4. PCI Configuration Access Registers

Offset	Register	Access	Reset	Section/Page
PCI Configuration Access Registers—Block Base Address 0x0_8300				
0x00	PCI_CONFIG_ADDRESS	W	All zeros	23.3.1.1/23-12
0x04	PCI_CONFIG_DATA	R/W	All zeros	23.3.1.2/23-14
0x08	PCI_INT_ACK	R	N/A	23.3.1.3/23-15
0x80–0xFF	Reserved	—	—	—

Table 23-5. PCI Memory-Mapped Registers

Offset	Register	Access	Reset	Section/Page
PCI Controller—Block Base Address 0x0_8500				
PCI Error Management Registers				
0x00	PCI error status register (PCI_ESR)	w1c	All zeros	23.3.2.1/23-15
0x04	PCI error capture disable register (PCI_ECDR)	R/W	All zeros	23.3.2.2/23-16
0x08	PCI error enable register (PCI_EER)	R/W	All zeros	23.3.2.3/23-17
0x0C	PCI error attributes capture register (PCI_EATCR)	R/W	All zeros	23.3.2.4/23-18
0x10	PCI error address capture register (PCI_EACR)	R	All zeros	23.3.2.5/23-19
0x14	PCI error extended address capture register (PCI_EEACR)	R	All zeros	23.3.2.6/23-20
0x18	PCI error data capture register (PCI_EDCR)	R/W	All zeros	23.3.2.7/23-20
PCI Control and Status Registers				
0x20	PCI general control register (PCI_GCR)	R/W	All zeros	23.3.2.8/23-20
0x24	PCI error control register (PCI_ECR)	R/W	All zeros	23.3.2.9/23-21

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 23-6 shows the bit settings of the PCI_CONFIG_ADDRESS register.

Table 23-6. 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.

Table 23-6. PCI_CONFIG_ADDRESS Field Descriptions (continued)

Bits	Name	Description																																																		
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.																																																		
		<table border="0"> <thead> <tr> <th>Value</th> <th>AD Signal that is Driving High</th> </tr> </thead> <tbody> <tr><td>01010</td><td>31</td></tr> <tr><td>01011</td><td>11</td></tr> <tr><td>01100</td><td>12</td></tr> <tr><td>01101</td><td>13</td></tr> <tr><td>01110</td><td>14</td></tr> <tr><td>01111</td><td>15</td></tr> <tr><td>10000</td><td>16</td></tr> <tr><td>10001</td><td>17</td></tr> <tr><td>10010</td><td>18</td></tr> <tr><td>10011</td><td>19</td></tr> <tr><td>10100</td><td>20</td></tr> <tr><td>10101</td><td>21</td></tr> <tr><td>10110</td><td>22</td></tr> <tr><td>10111</td><td>23</td></tr> <tr><td>11000</td><td>24</td></tr> <tr><td>11001</td><td>25</td></tr> <tr><td>11010</td><td>26</td></tr> <tr><td>11011</td><td>27</td></tr> <tr><td>11100</td><td>28</td></tr> <tr><td>11101</td><td>29</td></tr> <tr><td>11110</td><td>30</td></tr> <tr><td>11111</td><td>Special cycle / interrupt acknowledge</td></tr> <tr><td>00000</td><td>Internal access</td></tr> <tr><td>Others</td><td>Reserved</td></tr> </tbody> </table>	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
		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.																																																		
7–2	RN	Register number. Specifies the register being accessed in the PCI configuration space.																																																		
1–0	—	Reserved																																																		

23.3.1.2 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 23-4 shows the PCI_CONFIG_DATA register fields.

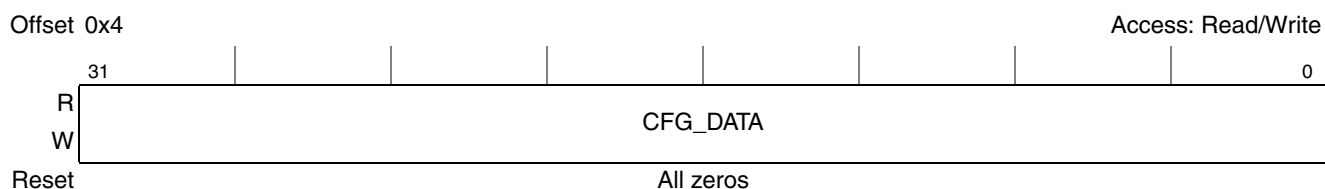


Figure 23-4. PCI_CONFIG_DATA

Table 23-7 shows the bit settings of the PCI_CONFIG_DATA register.

Table 23-7. 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.

23.3.1.3 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.

23.3.2 PCI Memory-Mapped Control and Status BE Registers

This section describes the control and status registers.

23.3.2.1 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 23-5 shows the PCI_ESR fields.

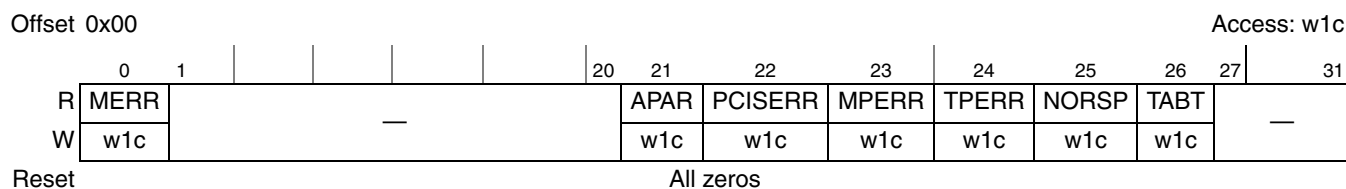


Figure 23-5. PCI Error Status Register (PCI_ESR)

Table 23-8 describes the bit settings of the PCI_ESR register.

Table 23-8. 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 23-3 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 $\overline{\text{PCI_DEVSEL}}$ assertion).
26	TABT	Target abort. Set when a PCI target abort occurs on a transaction initiated by this PCI controller.
27–31	—	Reserved

23.3.2.2 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 23-6 shows the PCI_ECDR fields.

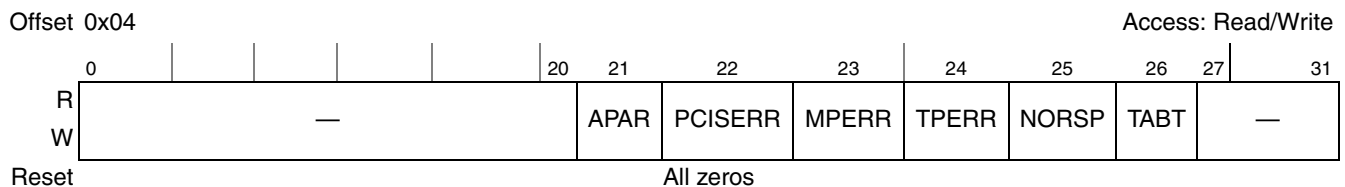


Figure 23-6. PCI Error Capture Disable Register (PCI_ECDR)

Table 23-10. PCI_EER Field Descriptions (continued)

Bits	Name	Description
26	TABT	Target abort. Generate an interrupt when the corresponding bit of the PCI_ESR is 1.
27–31	—	Reserved

23.3.2.4 PCI Error Attributes Capture Register (PCI_EATCR)

PCI_EATCR contains fields for storing information associated with the first PCI error captured. Figure 23-8 shows the PCI_EATCR fields.

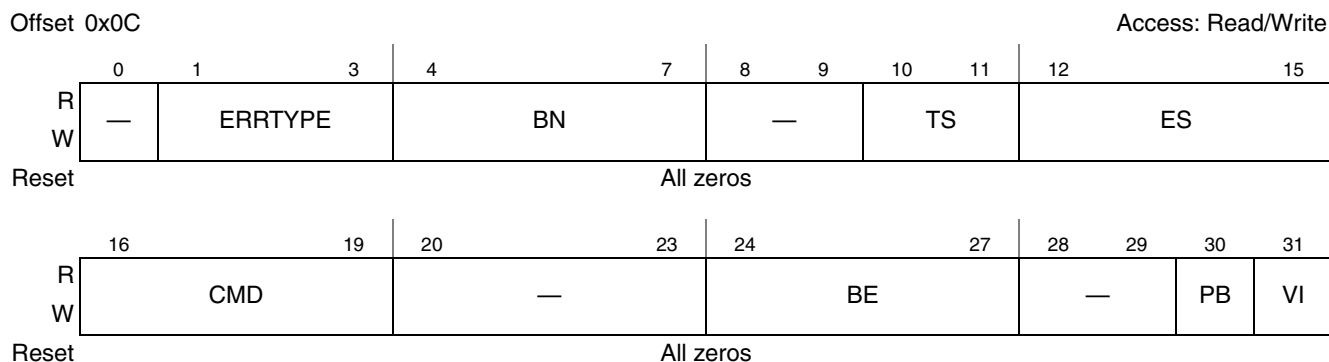


Figure 23-8. PCI Error Attributes Capture Register (PCI_EATCR)

Table 23-11 describes the bit settings of the PCI_EATCR register.

Table 23-11. 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
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

Table 23-11. PCI_EATCR Field Descriptions (continued)

Bits	Name	Description
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	—	Reserved
24–27	BE	PCI byte enables. Contains the PCI byte enables PCI_CBE[3:0] for the data word.
28–29	—	Reserved
30	PB	Parity bit. Contains the 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, and PCI_EDCR is valid. 0 No valid error information 1 Error information is valid

23.3.2.5 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 23-9](#) shows the PCI_EACR fields.

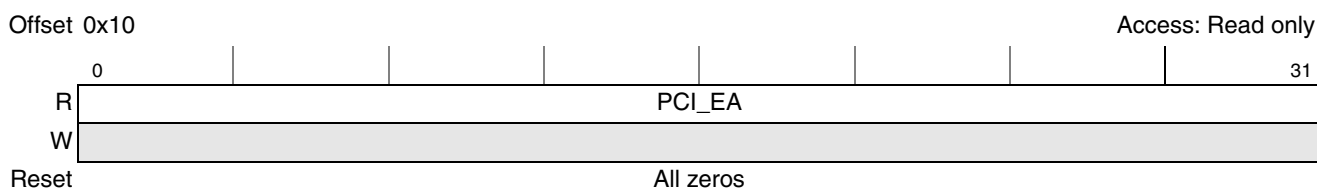


Figure 23-9. PCI Error Address Capture Register (PCI_EACR)

[Table 23-12](#) describes the bit settings of the PCI_EACR register.

Table 23-12. 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.

23.3.2.6 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 23-10](#) shows the PCI_EEACR fields.

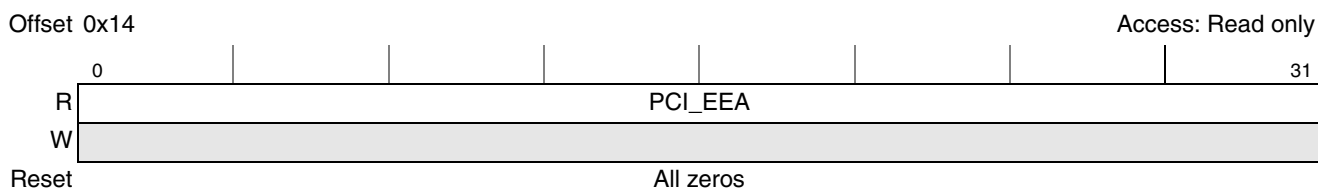


Figure 23-10. PCI Error Extended Address Capture Register (PCI_EEACR)

[Table 23-13](#) describes the bit settings of the PCI_EEACR register.

Table 23-13. 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.

23.3.2.7 PCI Error Data Capture Register (PCI_EDCR)

PCI_EDCR contains fields for storing the data associated with the first PCI error captured. [Figure 23-11](#) shows the PCI_EDCR fields.



Figure 23-11. PCI Error Data Capture Register (PCI_EDCR)

[Table 23-14](#) describes the bit settings of the PCI_EDCR register.

Table 23-14. PCI_EDCR Field Description

Bits	Name	Description
0–31	PCI_EDR	PCI error data. Contains the data associated with the first detected error.

23.3.2.8 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 23-12](#) shows the PCI_GCR fields.

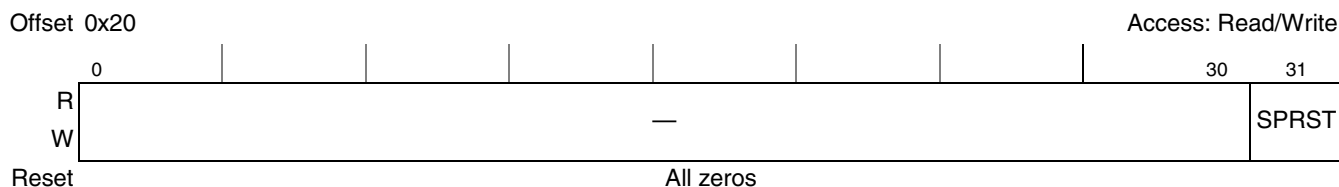


Figure 23-12. PCI General Control Register (PCI_GCR)

Table 23-15 shows the bit settings of PCI_GCR.

Table 23-15. PCI_GCR Field Descriptions

Bits	Name	Description
0–30	—	Reserved
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.

23.3.2.9 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 23-13 shows the PCI_ECR fields.

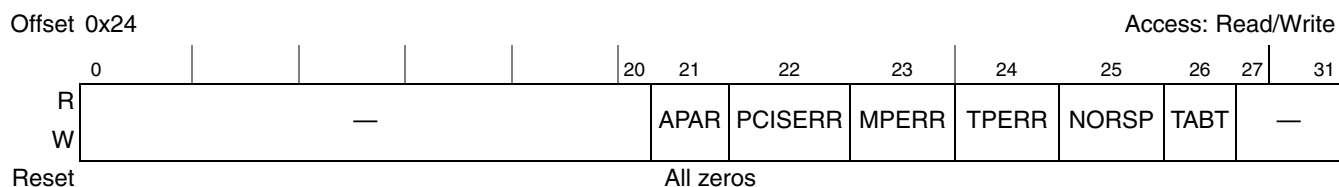


Figure 23-13. PCI Error Control Register (PCI_ECR)

Table 23-16 describes the bit settings of the PCI_ECR register.

Table 23-16. 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.

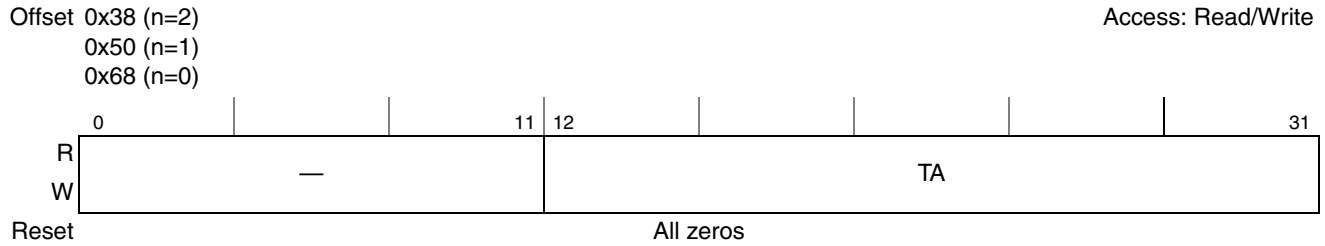


Figure 23-15. PCI Inbound Translation Address Registers (PITAR_n)

Table 23-18 shows the bit settings of PITAR_n.

Table 23-18. 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. The specified address must be aligned to the window size, as defined by PIWAR _n [IWS].

23.3.2.12 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 23-16 shows the PIBAR_x fields.

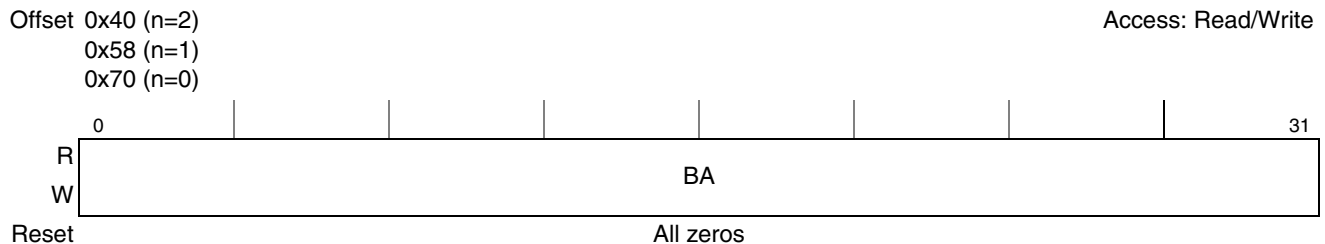


Figure 23-16. PCI Inbound Base Address Registers (PIBAR_n)

Table 23-19 shows the bit settings of PIBAR_n.

Table 23-19. 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 PIBAR ₀ , the upper 12 bits are reserved because only a 32-bit address is supported. The specified address must be aligned to the window size, as defined by PIWAR _n [IWS].

23.3.2.13 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 23-17 shows the PIEBAR_n fields.

PCI Bus Interface

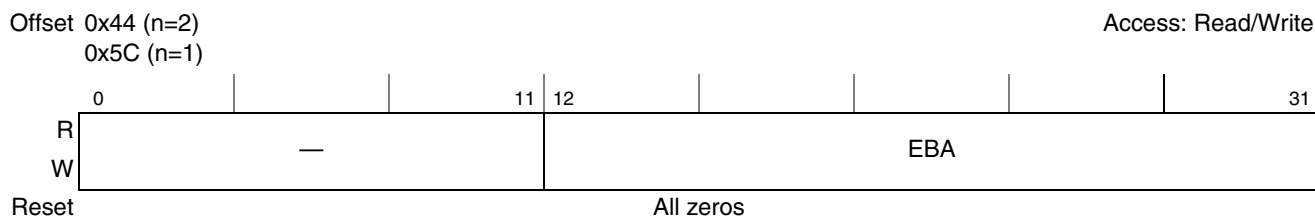


Figure 23-17. PCI Inbound Extended Base Address Registers (PIEBAR_n)

Table 23-20 shows the bit settings of PIEBAR_n.

Table 23-20. 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.

23.3.2.14 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 23-18 shows the PIWAR_n fields. See Section 4.3.1.1, “Reset Configuration Word Source,” for more information on reset configuration.

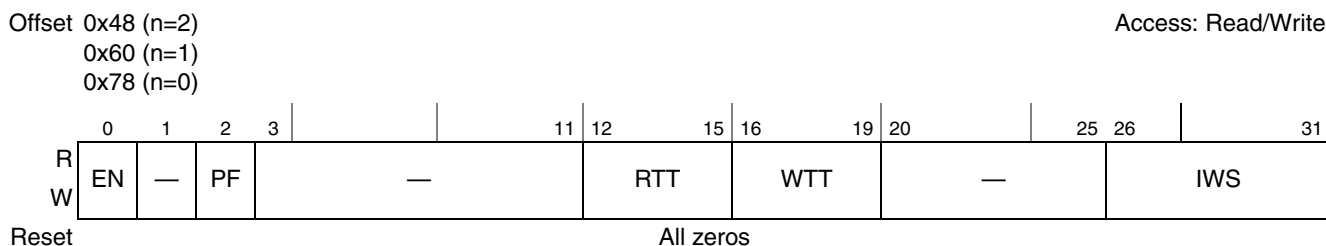


Figure 23-18. PCI Inbound Window Attribute Registers (PIWAR_n)

Table 23-21 shows the bit settings of PIWAR_n.

Table 23-21. 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
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

Table 23-21. PIWAR_n Field Descriptions (continued)

Bits	Name	Description
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

23.3.3 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 23-22 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 23-22. 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

Table 23-22. PCI Configuration Space Registers (continued)

Address	Use	Access
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

23.3.3.1 Vendor ID Configuration Register

Figure 23-19 shows the vendor ID fields. This is a read-only register.

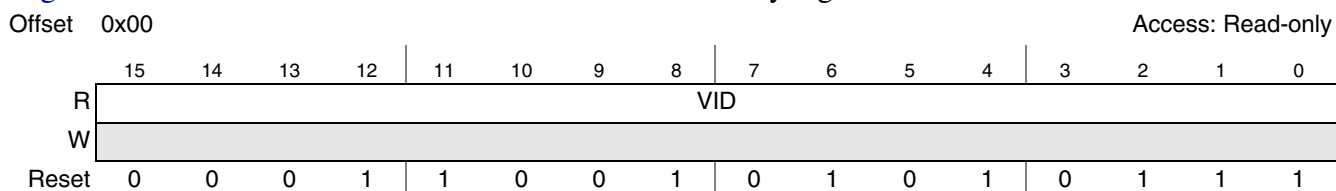


Figure 23-19. Vendor ID Configuration Register

Table 23-23 shows the bit settings of the vendor ID register.

Table 23-23. 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.

23.3.3.2 Device ID Configuration Register

Figure 23-20 shows the device ID fields. This is a read only register.

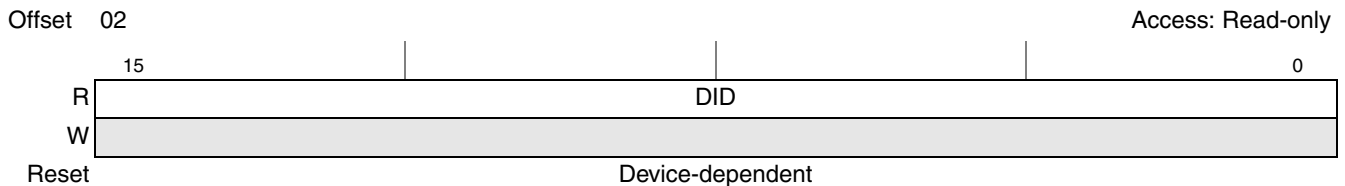


Figure 23-20. Device ID Configuration Register

Table 23-24 shows the bit settings of the device ID register.

Table 23-24. Device ID Configuration Register Field Descriptions

Bits	Name	Description
15–0	DID	Device ID. This field identifies the device. 0xC011 MPC8309

23.3.3.3 PCI Command Configuration Register

Figure 23-21 shows the PCI command fields.

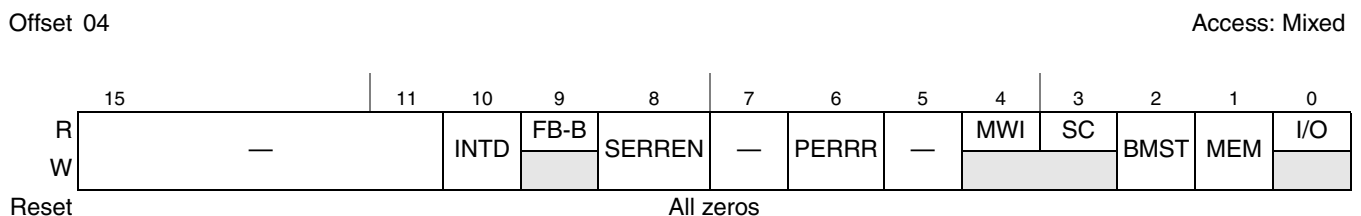


Figure 23-21. PCI Command Configuration Register

Table 23-25 shows the bit settings of the PCI command register.

Table 23-25. 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.

Table 23-25. PCI Command Configuration Register Field Descriptions (continued)

Bits	Name	Description
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.

23.3.3.4 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 23-22](#) 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

Figure 23-22. PCI Status Configuration Register

[Table 23-26](#) shows the bit settings of the PCI status register.

Table 23-26. 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.

Table 23-26. PCI Status Configuration Register Field Descriptions (continued)

Bits	Name	Description
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

23.3.3.5 Revision ID Configuration Register

Figure 23-23 shows the revision ID fields.

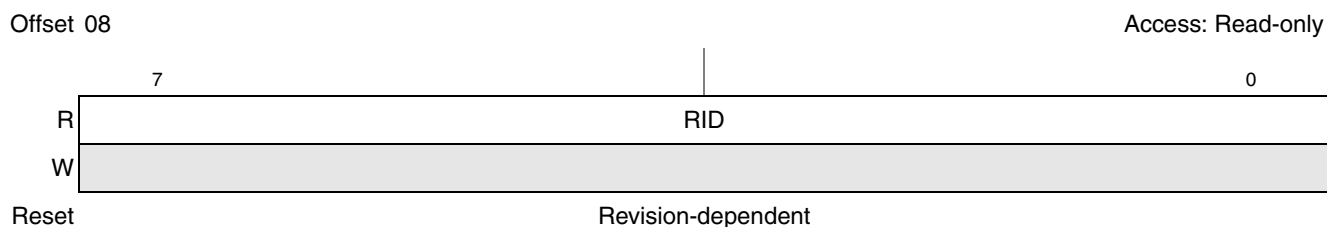

Figure 23-23. Revision ID Configuration Register

Table 23-27 shows the bit settings of the revision ID register.

Table 23-27. Revision ID Configuration Register Field Descriptions

Bits	Name	Description
7–0	RID	Revision ID. Specifies a revision code of the PCI controller. 0x10 means rev 1.0 and 0x11 means rev1.1 and others are reserved.

23.3.3.6 Standard Programming Interface Configuration Register

Figure 23-24 shows the standard programming interface fields. This is the lower byte of the class code.

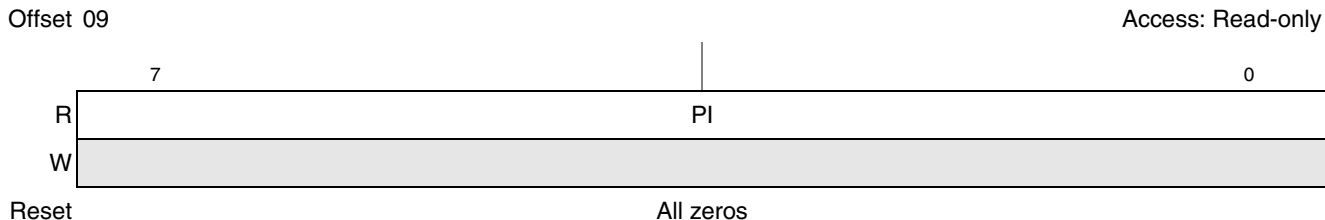


Figure 23-24. Standard Programming Interface Configuration Register

Table 23-28 shows the bit settings of the standard programming interface register.

Table 23-28. Standard Programming Interface Configuration Register Field Descriptions

Bits	Name	Description
7-0	PI	Programming interface. This field is hard-wired to 0x00.

23.3.3.7 Subclass Code Configuration Register

Figure 23-25 shows the subclass code fields. This is the middle byte of the class code.

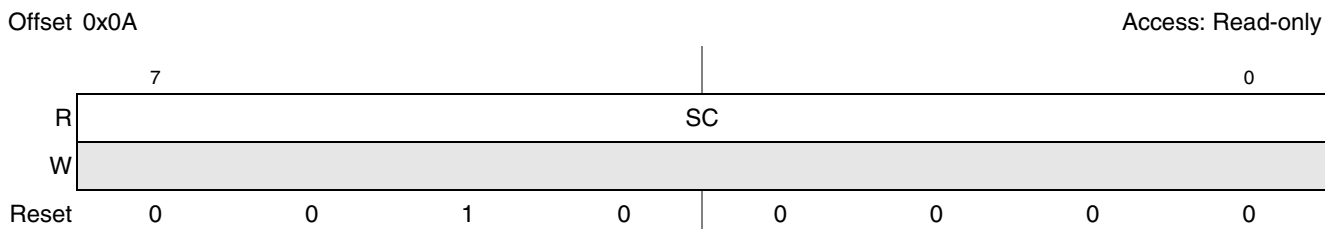


Figure 23-25. Subclass Code Configuration Register

Table 23-29 shows the bit settings of the subclass code register.

Table 23-29. 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.

23.3.3.8 Base Class Code Configuration Register

Figure 23-26 shows the base class code fields. This is the upper byte of the class code.

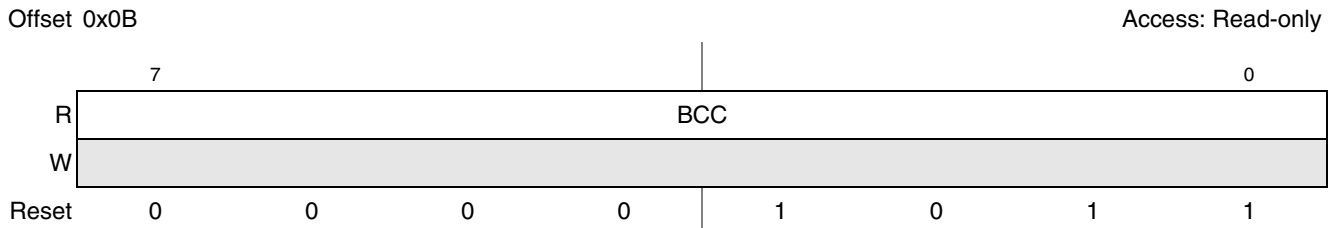


Figure 23-26. Base Class Code Configuration Register

Table 23-30 shows the bit settings of the class code register.

Table 23-30. 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.

23.3.3.9 Cache Line Size Configuration Register

Figure 23-27 shows the cache line size fields.

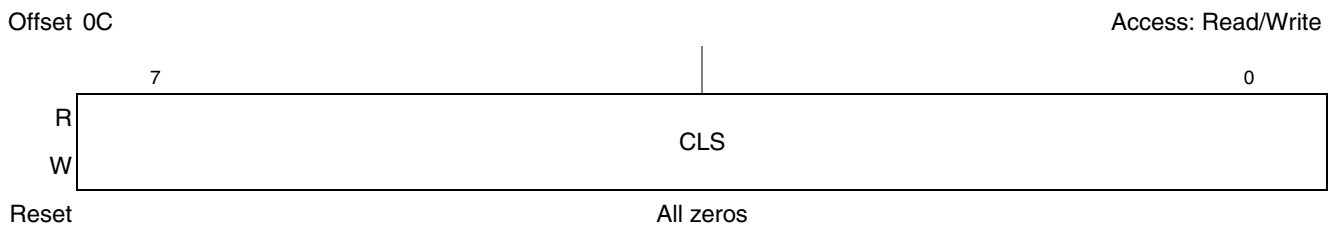


Figure 23-27. Cache Line Size Configuration Register

Table 23-31 shows the bit settings of the cache line size register.

Table 23-31. 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.

23.3.3.10 Latency Timer Configuration Register

Figure 23-28 shows the latency timer fields.

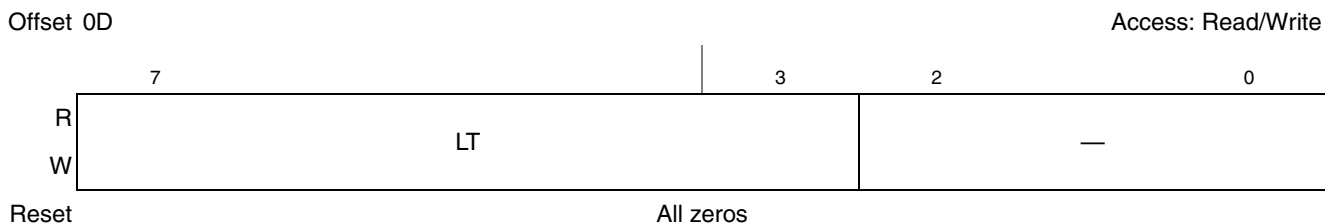


Figure 23-28. Latency Timer Configuration Register

Table 23-32 shows the bit settings of the latency timer register.

Table 23-32. 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

23.3.3.11 Header Type Configuration Register

Figure 23-29 shows the read-only header type register, which is hard-wired to 0x00.

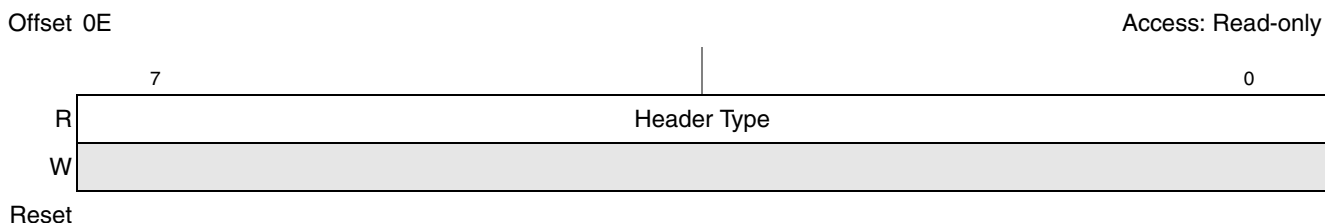


Figure 23-29. Header Type Configuration Register

23.3.3.12 BIST Control Configuration Register

Figure 23-30 shows the read-only BIST control register, which is hard-wired to 0x00.

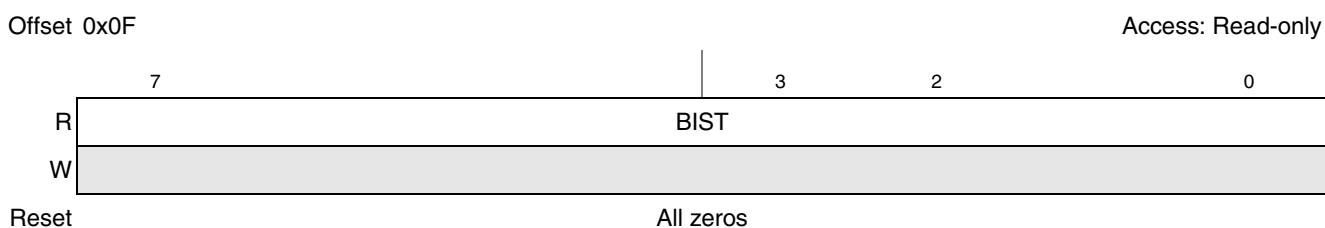


Figure 23-30. BIST Control Configuration Register

23.3.3.13 PIMMR Base Address Configuration Register

Figure 23-31 shows the PIMMR base address register fields.

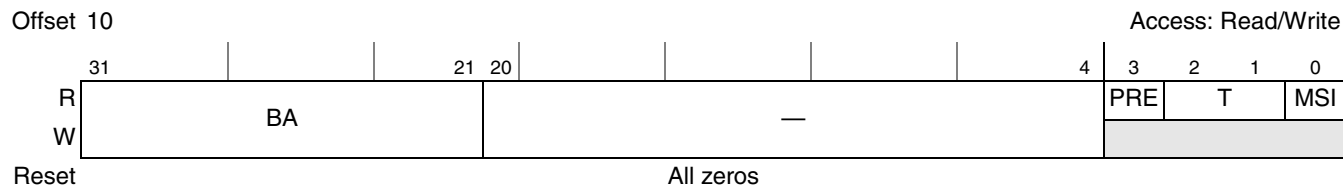


Figure 23-31. PIMMR Base Address Configuration Register

Table 23-33 shows the bit settings of the PIMMR base address register.

Table 23-33. PIMMR Base Address Configuration Register Field Descriptions

Bits	Name	Description
31–21	BA	Base address. Defines the base address for the internal (on-chip) memory-mapped register space. The size of this space is 2 Mbytes.
20–4	—	Reserved
3	PRE	Prefetchable. Hard-wired to 0.
2–1	T	Type. Hard-wired to 00.
0	MSI	Memory space indicator. Hard-wired to 0

23.3.3.14 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 23-32 shows the GPL base address register 0 fields.



Figure 23-32. GPL Base Address Register 0

Table 23-34 shows the bit settings of the GPL base address register 0.

write operation does not change bits that are masked by the IWS field. For read operations these masked bits will always return zeros.

Figure 23-34 shows the GPL extended base address registers 1–2 fields.

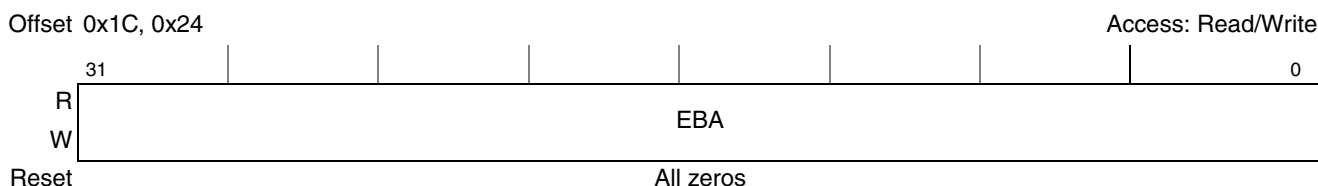


Figure 23-34. GPL Extended Base Address Registers 1–2

Table 23-36 shows the bit settings of the GPL extended base address register 1–2.

Table 23-36. 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.

23.3.3.17 Subsystem Vendor ID Configuration Register

Figure 23-35 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.

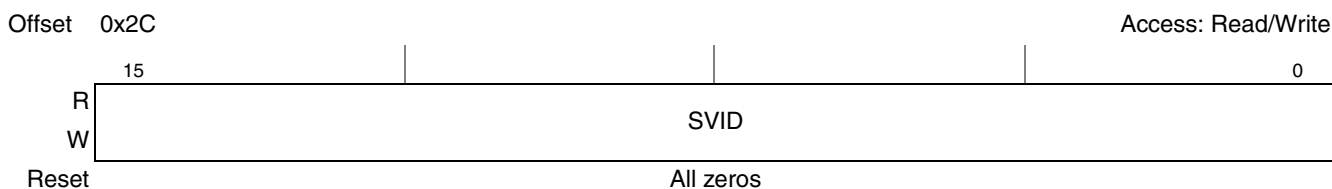


Figure 23-35. Subsystem Vendor ID Configuration Register

Table 23-37 shows the bit settings of the subsystem vendor ID configuration register.

Table 23-37. 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.

23.3.3.18 Subsystem Device ID Configuration Register

Figure 23-36 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.

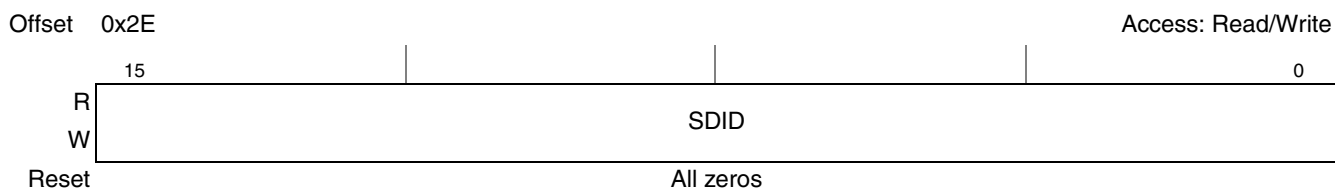


Figure 23-36. Subsystem Device ID Configuration Register

Table 23-38 shows the bit settings of the subsystem device ID configuration register.

Table 23-38. 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.

23.3.3.19 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 23-37 shows the capabilities pointer configuration register fields.

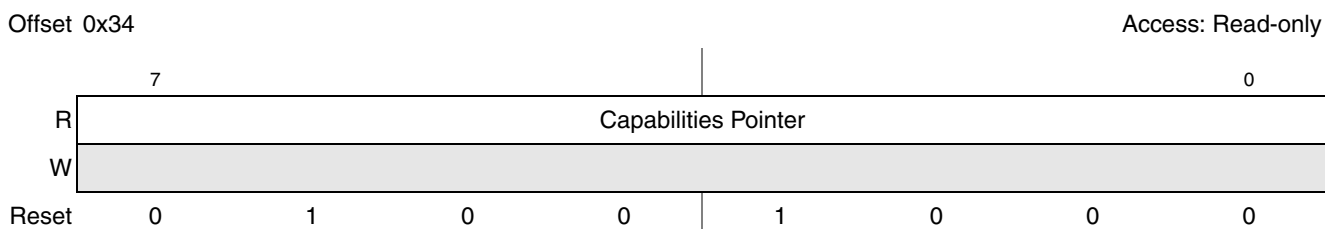


Figure 23-37. Capabilities Pointer Configuration Register

23.3.3.20 Interrupt Line Configuration Register

Figure 23-38 shows the interrupt line configuration register fields.

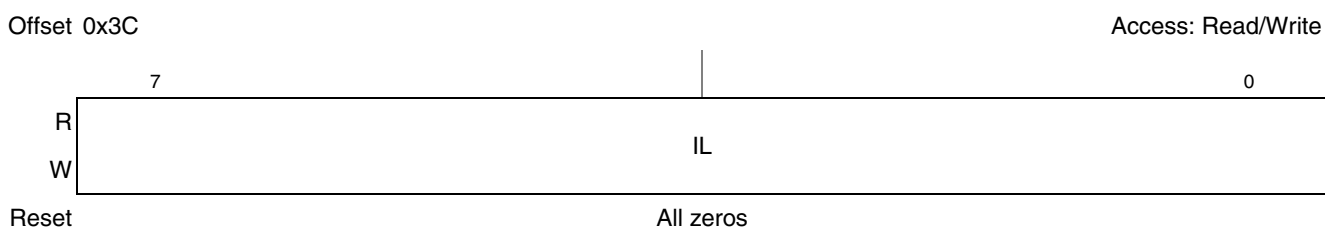


Figure 23-38. Interrupt Line Configuration Register

Table 23-39 shows the bit settings of the interrupt line configuration register.

Table 23-39. 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.

23.3.3.21 Interrupt Pin Configuration Register

The interrupt pin configuration register tells which interrupt pin is used (0x01 means PCI_INTA).

Figure 23-39 shows the interrupt pin configuration register fields.

Offset 0x3D

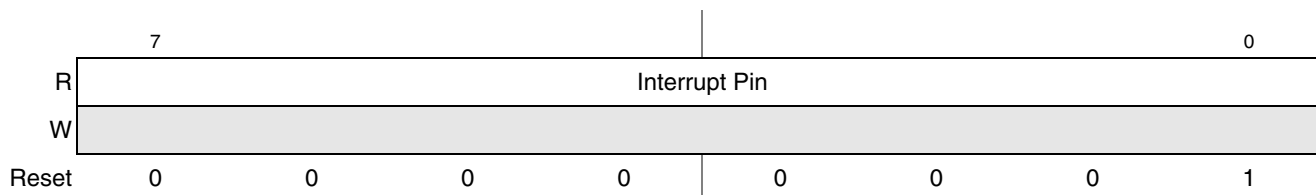


Figure 23-39. Interrupt Pin Register

23.3.3.22 Minimum Grant Configuration Register

Figure 23-40 shows the minimum grant configuration register fields.

Offset 0x3E

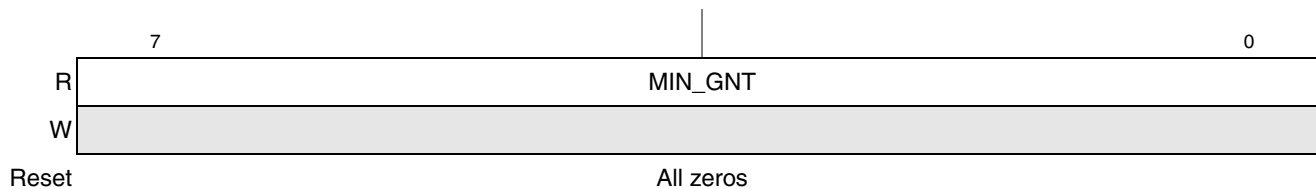


Figure 23-40. Minimum Grant Configuration Register

23.3.3.23 Maximum Latency Configuration Register

Figure 23-41 shows the maximum latency configuration register fields.

Offset 0x3F

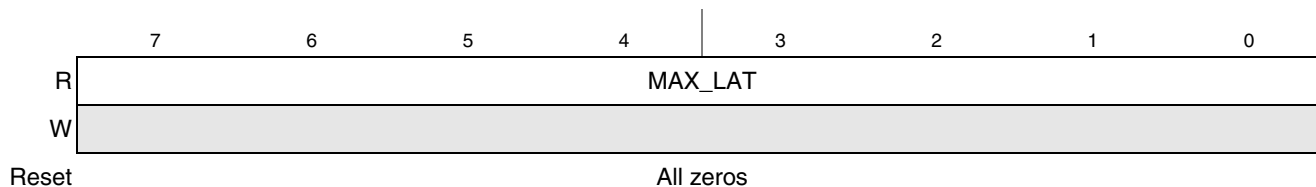


Figure 23-41. Maximum Latency Configuration Register

23.3.3.24 PCI Function Configuration Register

Figure 23-42 shows the PCI function configuration register fields.

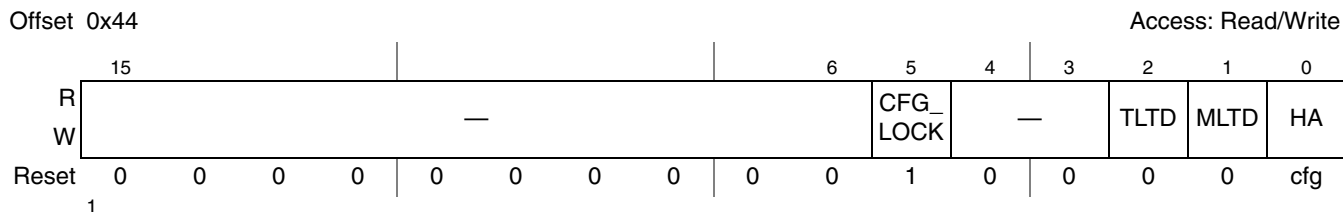


Figure 23-42. PCI Function Configuration Register

Table 23-40 shows the bit settings of the PCI function configuration register.

Table 23-40. 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,” for more information on reset configuration.
4–3	—	Reserved
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.
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. See Section 4.3.1.1, “Reset Configuration Word Source,” for more information on reset configuration. 0 Host mode 1 Agent mode

23.3.3.25 PCI Arbiter Control Register (PCIACR)

Figure 23-43 shows the PCI arbiter control register (PCIACR) fields.

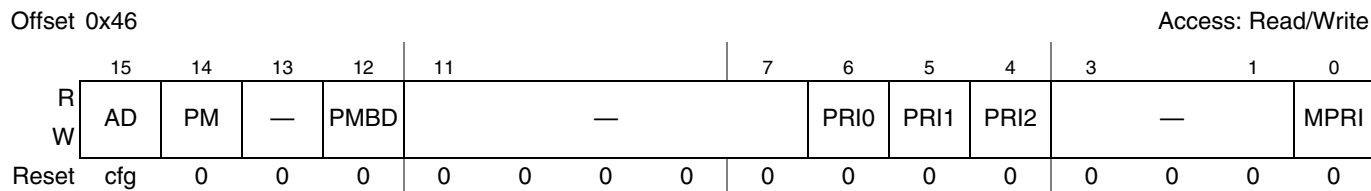


Figure 23-43. PCI Arbiter Control Register (PCIACR)

Table 23-41 shows the bit settings of the PCIACR.

Table 23-41. PCI Arbiter Control Register (PCIACR) Field Descriptions

Bits	Name	Description
15	AD	Arbiter disable. Indicates whether the PCI controller's Arbiter function is enabled or disabled. See Table 4-11., "Reset Configuration Word High Bit Settings," 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
6–4	PRI _n	Priority level for master <i>n</i> . When the PCI controller functions as the arbiter for the PCI bus, each PRI _n bit determines the arbitration priority level for the PCI master connected to the REQ _n /GNT _n pair. 0 Low priority 1 High priority
3–1	—	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

23.3.3.26 Hot Swap Register Block

Figure 23-44 shows the hot swap register block fields.

Offset 0x48

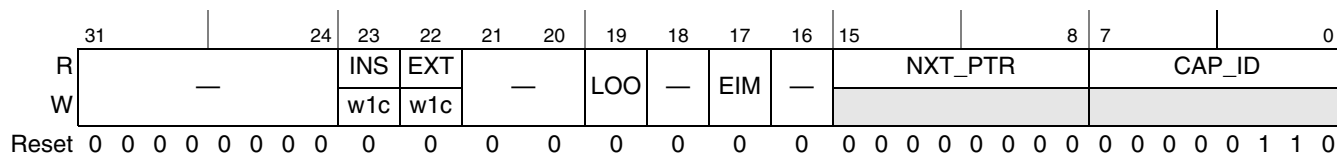


Figure 23-44. Hot Swap Register Block

Table 23-42 shows the bit settings of the Hot Swap register block.

Table 23-42. 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, end of capabilities list.
7–0	CAP_ID	Capability ID for hot swap (hardwired to 0x06)

23.4 Functional Description

The following sections discuss the operation of the PCI controller.

23.4.1 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{REQn}) output and grant (\overline{GNTn}) 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).

The PCI internal arbiter supports three external masters (besides the PCI controller itself) by using the \overline{REQ} signals and generating the \overline{GNT} signals.

During reset, the PCI controller samples the reset configuration bit (and programs the AD 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 AD bit in the arbiter configuration register (see [Section 23.3.3.25, “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.

23.4.1.1 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 23.3.3.25, “PCI Arbiter Control Register \(PCIACR\),”](#) for more information).

23.4.1.2 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 23.3.3.25, “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 $(N+1) \times M$ 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 23-45](#). 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.

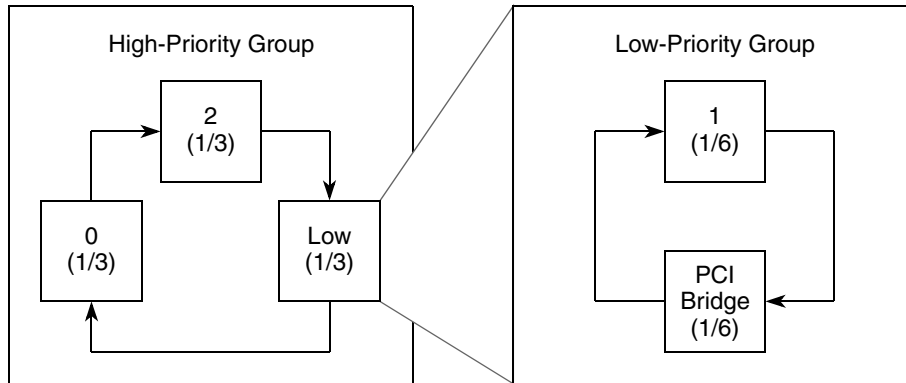


Figure 23-45. PCI Arbitration Example

23.4.1.3 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.

23.4.1.4 Master Latency Timer

The PCI controller implements the master latency timer register (see [Section 23.3.3.10, “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 $\overline{\text{PCI_GNT}}$ signals. If the $\overline{\text{PCI_GNT}}$ 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 23.3.3.24, “PCI Function Configuration Register,”](#) for more information).

23.4.2 Bus Commands

PCI bus commands indicate the type of transaction occurring on the bus. These commands are encoded on $\text{PCI_C}/\overline{\text{BE}}[3:0]$ during the address phase of the transaction. PCI bus commands are described in

Table 23-43.

Table 23-43. 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 23.4.4.6, "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.
0b0111	Memory write	Yes	Yes	Accesses agents mapped in memory address space. Note that for inbound writes less than 4-bytes, the PCI controller splits the transaction into single byte writes to the target. Thus, the PCI interface cannot be used to perform single beat writes to 16-bit devices on the local peripheral interfaces.
0b100x	—	—	—	Reserved. No response occurs.
0b1010	Configuration read	Yes	Yes	As a host, accesses the configuration space of each agent. An agent is selected when its IDSEL signal is asserted. See Section 23.4.4.4, "Host Mode Configuration Access," for more information on configuration accesses. As an agent, a configuration read is accepted if the IDSEL input is asserted during the address phase of the PCI configuration read cycle.
0b1011	Configuration write	Yes	Yes	As a host, accesses the configuration space of each agent. An agent is selected when its IDSEL signal is asserted. See Section 23.4.4.4, "Host Mode Configuration Access," for more information on configuration accesses. As an agent, a configuration write is accepted if the IDSEL input is asserted during the address phase of the PCI configuration write cycle.
0b1100	Memory read multiple	Yes	Yes	Initiator has intention to perform a read from prefetchable space that will cross cacheline boundary. So, when acts as a target, this PCI controller will prefetch data from next cacheline while it is returning the fetched data of the current cacheline.
0b1101	Dual address cycle	No	Yes	Transfers an 8-byte address to devices.
0b1110	Memory read line	Yes	Yes	Initiator has intention to perform a read, from prefetchable space, till to the end of the cacheline. So, when acts as a target, this PCI controller will prefetch data till to the end of this cacheline.
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.

23.4.3 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.

23.4.3.1 Basic Transfer Control

PCI data transfers are controlled by the following signals:

- $\overline{\text{PCI_FRAME}}$ is driven by an initiator to indicate the beginning and end of a transaction.
- $\overline{\text{PCI_IRDY}}$ (initiator ready) is driven by an initiator, allowing it to force wait cycles.
- $\overline{\text{PCI_TRDY}}$ (target ready) is driven by a target, allowing it to force wait cycles.

The bus is idle when both $\overline{\text{PCI_FRAME}}$ and $\overline{\text{PCI_IRDY}}$ are negated. The first clock cycle in which $\overline{\text{PCI_FRAME}}$ 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 $\overline{\text{PCI_IRDY}}$ and $\overline{\text{PCI_TRDY}}$ are asserted. Once the PCI controller, as an initiator, has asserted $\overline{\text{PCI_IRDY}}$, it does not change $\overline{\text{PCI_IRDY}}$ or $\overline{\text{PCI_FRAME}}$ until the current data phase completes, regardless of the state of $\overline{\text{PCI_TRDY}}$. Once the PCI controller, as a target, has asserted $\overline{\text{PCI_TRDY}}$ or $\overline{\text{PCI_STOP}}$ it does not change $\overline{\text{PCI_DEVSEL}}$, $\overline{\text{PCI_TRDY}}$, or $\overline{\text{PCI_STOP}}$ until the current data phase completes.

When the PCI controller (as a master) intends to complete only one more data transfer, $\overline{\text{PCI_FRAME}}$ is negated and $\overline{\text{PCI_IRDY}}$ is asserted (or kept asserted) indicating the initiator is ready. After the target indicates it is ready ($\overline{\text{PCI_TRDY}}$ asserted) the bus returns to the idle state.

23.4.3.2 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 $\overline{\text{PCI_DEVSEL}}$ 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 23.4.3.6, “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/ $\overline{\text{BE}}$ [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.

23.4.3.3 Device Selection

As a target, the PCI controller drives $\overline{\text{PCI_DEVSEL}}$ one clock following the address phase as indicated in the configuration space status register; see [Section 23.3.3.4, “PCI Status Configuration Register,”](#) for more information. The PCI controller as a target qualifies the address/data lines with $\overline{\text{PCI_FRAME}}$ before asserting $\overline{\text{PCI_DEVSEL}}$. The $\overline{\text{PCI_DEVSEL}}$ signal is asserted at or before the clock edge at which the PCI controller enables its $\overline{\text{PCI_TRDY}}$, $\overline{\text{PCI_STOP}}$, or data (for a read). The $\overline{\text{PCI_DEVSEL}}$ signal is not negated until $\overline{\text{PCI_FRAME}}$ is negated, with $\overline{\text{PCI_IRDY}}$ asserted and either $\overline{\text{PCI_STOP}}$ or $\overline{\text{PCI_TRDY}}$ asserted. The exception to this is a target-abort; see [Section 23.4.3.8, “Transaction Termination,”](#) for more information.

As an initiator, if the PCI controller does not see the assertion of $\overline{\text{PCI_DEVSEL}}$ within 4 clocks of $\overline{\text{PCI_FRAME}}$, it terminates the transaction with a master-abort as described in [Section 23.4.3.8, “Transaction Termination,”](#) for more information.

23.4.3.4 Byte Enable Signals

The byte enable signals ($\overline{\text{BE}}$ [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.

23.4.3.5 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. $\overline{\text{PCI_IRDY}}$, $\overline{\text{PCI_TRDY}}$, and $\overline{\text{PCI_DEVSEL}}$ use the address phase as their turnaround-cycle. $\overline{\text{PCI_FRAME}}$, $\overline{\text{PCI_C/BE}}$ [3:0], and AD[31:0] use the idle cycle between transactions as their turnaround-cycle. (An idle cycle in PCI is when both $\overline{\text{PCI_FRAME}}$ and $\overline{\text{PCI_IRDY}}$ are negated).

Byte lanes not involved in the current data transfer are driven to a stable condition even though the data is not valid.

23.4.3.6 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.

23.4.3.7 Read and Write Transactions

Both read and write transactions begin with an address phase followed by a data phase. The address phase occurs when $\overline{\text{PCI_FRAME}}$ is asserted for the first time, and the AD[31:0] signals contain a byte address and the PCI_C/ $\overline{\text{BE}}$ [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 $\overline{\text{PCI_FRAME}}$ is asserted for the first time and the PCI_C/ $\overline{\text{BE}}$ [3:0] signals indicate a read command. [Figure 23-46](#) shows an example of a single beat read transaction.

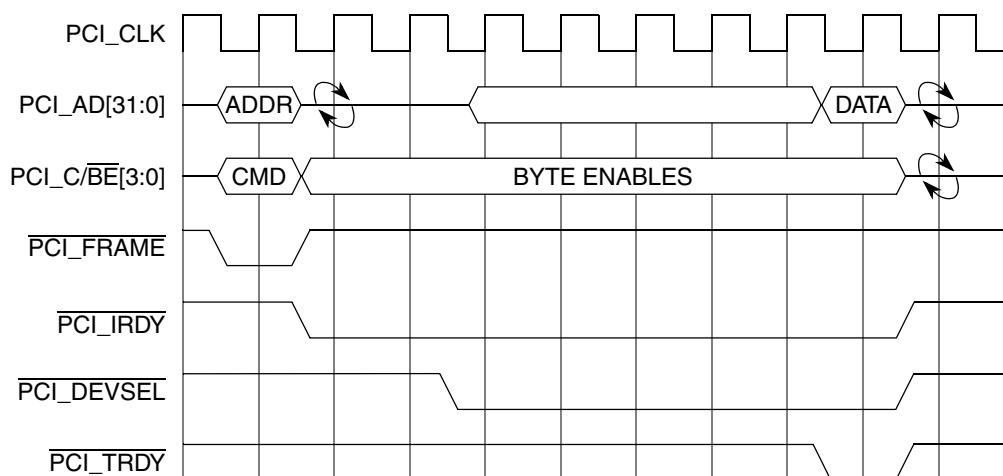


Figure 23-46. Single Beat Read Example

Figure 23-47 shows an example of a burst read transaction.

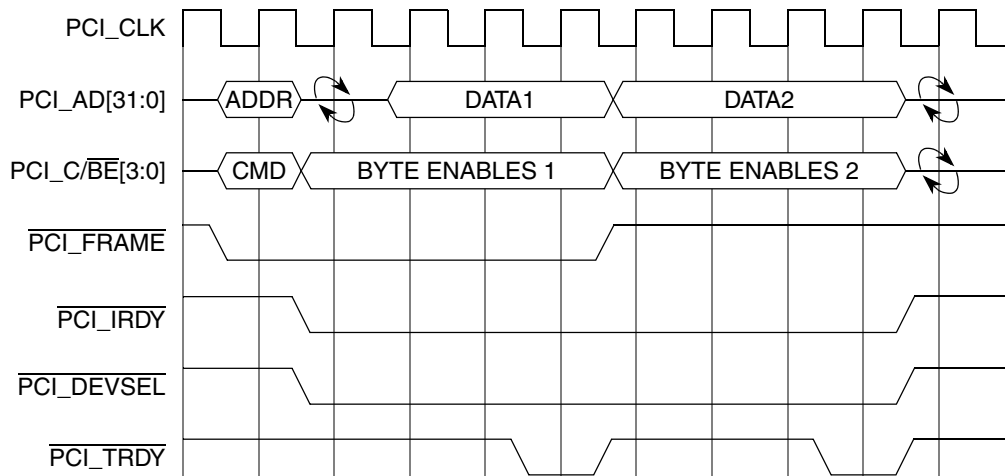


Figure 23-47. Burst Read Example

During the turnaround-cycle following the address phase, the $\overline{\text{PCI_C/BE}}[3:0]$ signals indicate which byte lanes are involved in the data phase. The turnaround-cycle must be enforced by the target with the $\overline{\text{PCI_TRDY}}$ signal if using fast $\overline{\text{PCI_DEVSEL}}$ assertion. The earliest the target can provide valid data is one cycle after the turnaround cycle. The target must drive the $\text{AD}[31:0]$ signals when $\overline{\text{PCI_DEVSEL}}$ is asserted except during the turnaround cycle.

The data phase completes when data is transferred, which occurs when both $\overline{\text{PCI_IRDY}}$ and $\overline{\text{PCI_TRDY}}$ are asserted on the same clock edge. When either is negated, a wait cycle is inserted and no data is transferred. To indicate the last data phase $\overline{\text{PCI_IRDY}}$ must be asserted when $\overline{\text{PCI_FRAME}}$ is negated.

A write transaction starts when $\overline{\text{PCI_FRAME}}$ is asserted for the first time and the $\overline{\text{PCI_C/BE}}[3:0]$ signals indicate a write command. Figure 23-48 shows an example of a single-beat write transaction.

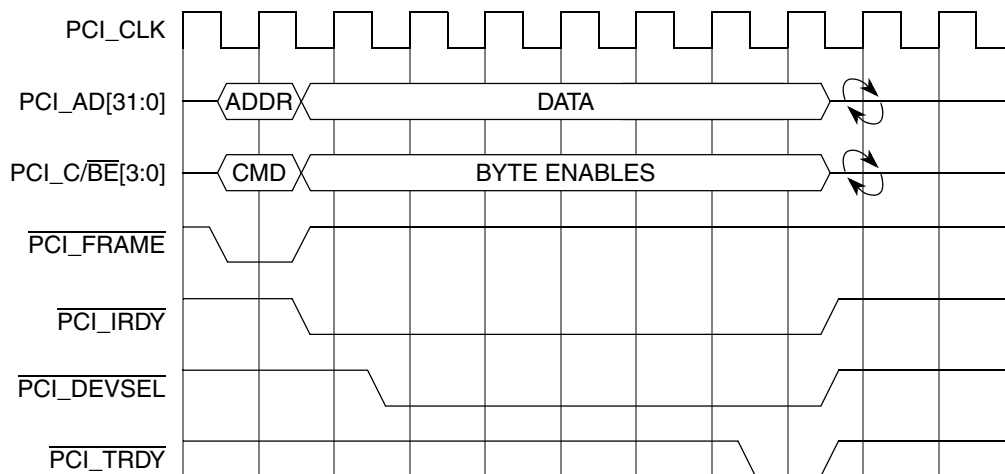


Figure 23-48. Single Beat Write Example

Figure 23-49 shows an example of a burst write transaction.

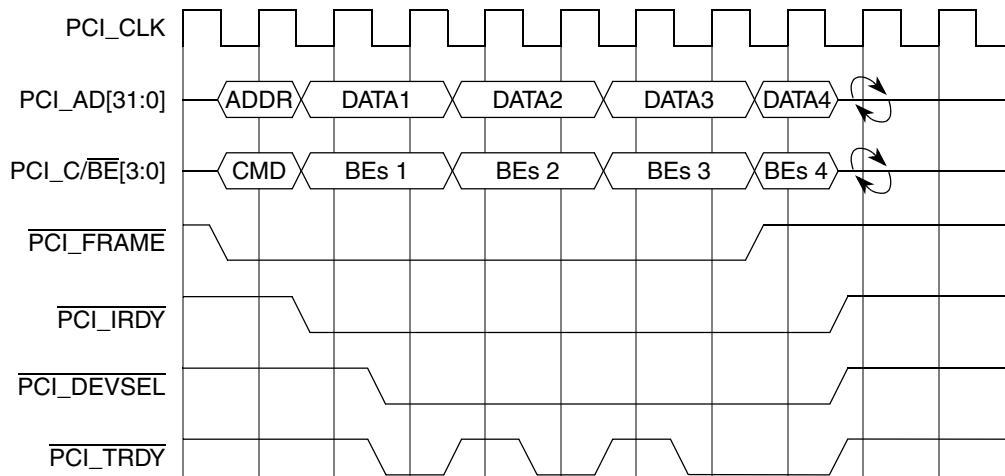


Figure 23-49. Burst Write Example

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.

23.4.3.8 Transaction Termination

The termination of a PCI transaction is orderly and systematic, regardless of the cause of the termination. All transactions end when $\overline{\text{PCI_FRAME}}$ and $\overline{\text{PCI_IRDY}}$ are both negated, indicating the idle cycle.

The PCI controller as an initiator terminates a transaction when $\overline{\text{PCI_FRAME}}$ is negated and $\overline{\text{PCI_IRDY}}$ is asserted. This indicates that the final data phase is in progress. The final data transfer occurs when both $\overline{\text{PCI_TRDY}}$ and $\overline{\text{PCI_IRDY}}$ are asserted. A master-abort is an abnormal case of a master initiated termination. If the PCI controller detects that $\overline{\text{PCI_DEVSEL}}$ has remained negated for more than four clocks after the assertion of $\overline{\text{PCI_FRAME}}$, it negates $\overline{\text{PCI_FRAME}}$ and then, on the next clock, negates $\overline{\text{PCI_IRDY}}$. 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 $\overline{\text{PCI_STOP}}$. Once asserted, $\overline{\text{PCI_STOP}}$ remains asserted until $\overline{\text{PCI_FRAME}}$ is negated. Depending on the circumstances, data may or may not be transferred during the request for termination. If $\overline{\text{PCI_TRDY}}$ and $\overline{\text{PCI_IRDY}}$ are asserted during the assertion of $\overline{\text{PCI_STOP}}$, data is transferred. This type of target-initiated termination is called a disconnect B, shown in Figure 23-50. If $\overline{\text{PCI_TRDY}}$ is asserted when $\overline{\text{PCI_STOP}}$ is asserted but $\overline{\text{PCI_IRDY}}$ is not, $\overline{\text{PCI_TRDY}}$ must remain asserted until $\overline{\text{PCI_IRDY}}$ is asserted and the data is transferred. This is called a disconnect A target-initiated termination, also shown in Figure 23-50. However, if $\overline{\text{PCI_TRDY}}$ is negated when $\overline{\text{PCI_STOP}}$ 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 23-50).

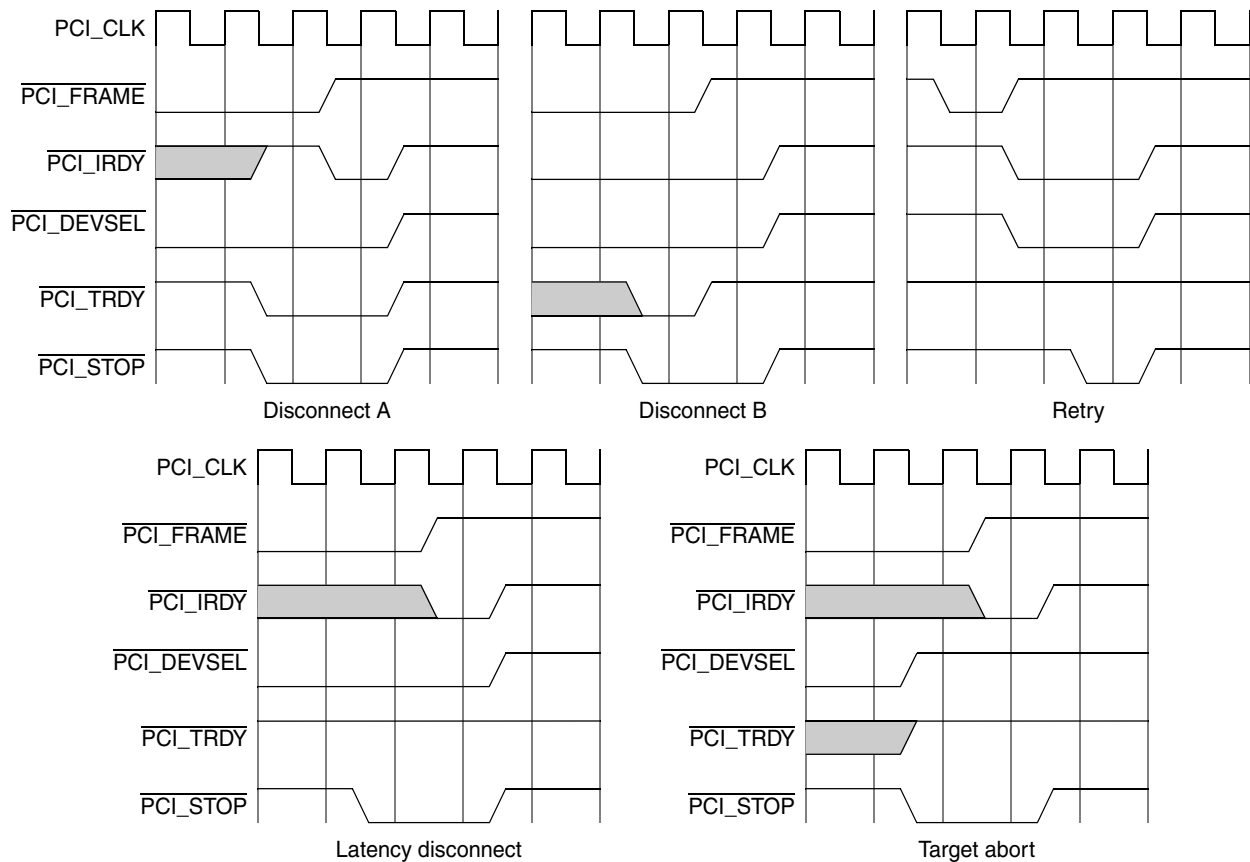


Figure 23-50. Target-Initiated Terminations

Note that when an initiator is terminated by $\overline{\text{PCI_STOP}}$, it must negate its $\overline{\text{REQn}}$ 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 $\overline{\text{REQn}}$ immediately following the two clocks or potential starvation may occur. If the initiator does not intend to complete the transaction, it can assert $\overline{\text{REQn}}$ 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 23-50](#)).
- 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 23.3.3.24, “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 23.3.3.24, “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 23-48](#).

Note that because a target can determine whether or not data is transferred (when both $\overline{\text{PCI_IRDY}}$ and $\overline{\text{PCI_TRDY}}$ are asserted), if it wants to do only one more data transfer and then stop, it may assert $\overline{\text{PCI_TRDY}}$ and $\overline{\text{PCI_STOP}}$ 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 $\overline{\text{PCI_STOP}}$ is asserted and $\overline{\text{PCI_DEVSEL}}$ 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 23-48](#).

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.

23.4.4 Other Bus Operations

The following sections provide information on additional PCI bus operations.

23.4.4.1 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 $\overline{\text{PCI_DEVSEL}}$ in the previous cycle, it delays the assertion of $\overline{\text{PCI_DEVSEL}}$ and $\overline{\text{PCI_TRDY}}$ for one cycle to allow the other target to get off the bus.

23.4.4.2 Dual Address Cycles

The PCI controller supports dual address cycle (DAC) commands (64-bit addressing on PCI bus) as a target only. 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.

23.4.4.3 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 23.3.2.14, “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.

23.4.4.4 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 23.4.4.6, “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_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 $\overline{\text{PCI_FRAME}}$.

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 23.3.2.9, “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 23.3.2.3, “PCI Error Enable Register \(PCI_EER\).”](#)

23.4.4.5 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.

23.4.4.6 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 $\overline{\text{PCI_DEVSEL}}$ 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 $\overline{\text{PCI_FRAME}}$ and completes when

$\overline{\text{PCI_FRAME}}$ and $\overline{\text{PCI_IRDY}}$ 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 $\overline{\text{PCI_IRDY}}$ 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.

23.4.4.7 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_PAR`). During the data phase, the byte enable signals determine which bytes are involved in the transaction. The interrupt vector must be returned when $\overline{\text{PCI_TRDY}}$ is asserted.

An interrupt acknowledge transaction can also be issued on the PCI bus by reading from the `PCI_INT_ACK` register.

23.4.5 Error Functions

This section describes PCI bus errors.

23.4.5.1 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_PAR is generated such that the number of ones on PCI_AD[31:0], PCI_C/BE[3:0] and PCI_PAR equals an even number. The PCI_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 PCI_FRAME) and for valid data transfers (PCI_IRDY and PCI_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 23.3.3.4, “PCI Status Configuration Register.”)

23.4.5.2 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 23.3.3.3, “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 PCI_PERR two clocks after the actual data transfer in which a data parity error is detected, and keeps PCI_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 PCI_PERR.

Figure 23-51 shows the possible assertion points for PCI_PERR if the PCI controller detects a data parity error.

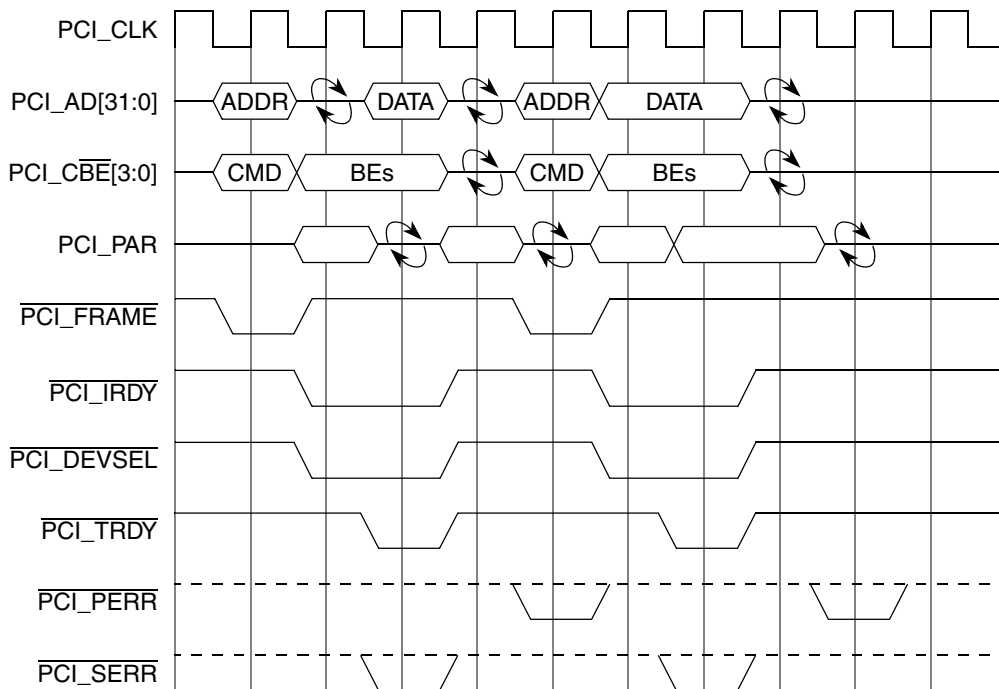


Figure 23-51. PCI Parity Operation

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 $\overline{\text{PCI_SERR}}$, 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 $\overline{\text{PCI_SERR}}$ is conditioned on the parity-error-response bit being enabled in the command register. $\overline{\text{PCI_SERR}}$ 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 $\overline{\text{MCP}}$. [Figure 23-51](#) shows where the PCI controller could detect an address parity error and assert $\overline{\text{PCI_SERR}}$ or where the PCI controller, acting as an initiator, checks for the assertion of $\overline{\text{PCI_SERR}}$ signaled by the target detecting an address parity error.

As a target that asserts $\overline{\text{PCI_SERR}}$ 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 $\overline{\text{PCI_PERR}}$ 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.

23.4.6 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 23-52](#) shows an example translation window for inbound memory accesses.

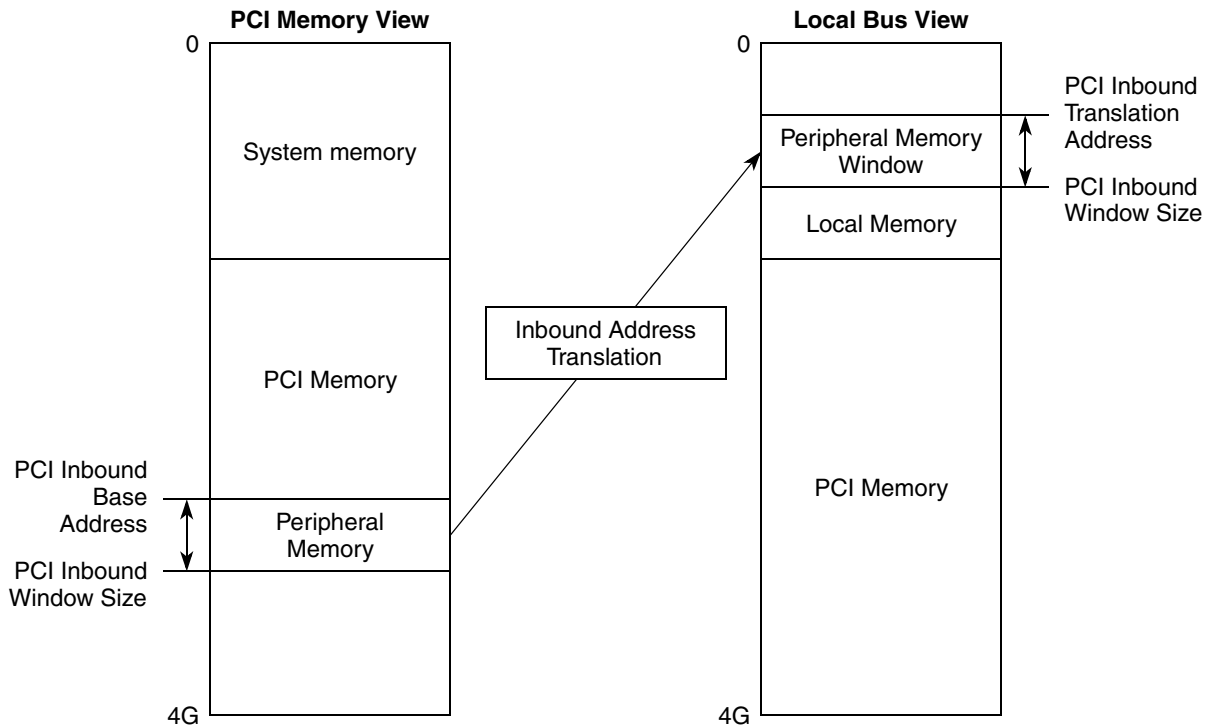


Figure 23-52. Inbound PCI Memory Address Translation

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 `PCI_DEVSEL` until the inbound translation windows are enabled.

23.4.7 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.

23.4.8 Byte Ordering

Whenever data must cross a bridge between two busses, the byte ordering of data on the source and destination buses must be considered. The internal platform bus of this device is inherently big endian and the PCI bus interface is inherently little endian.

There are two methods to handle ordering of data as it crosses a bridge—address invariance and data invariance. Address invariance preserves the addressing of bytes within a scalar data element, but not the relative significance of the bytes within that scalar. Conversely, data invariance preserves the relative significance of bytes within a scalar, but not the addressing of the individual bytes that make up a scalar.

This device uses address invariance as its byte ordering policy

As stated above, address invariance preserves the byte address of each byte on an I/O interface as it is placed in memory or moved into a register. This policy can have the effect of reversing the significance order of bytes (most significant to least significant and vice versa), but it has the benefit of preserving the format of general data structures. Provided that software is aware of the endianness and format of the data structure, it can correctly interpret the data on either side of the bridge.

Figure 23-53 shows the transfer of a 4-byte scalar, 0x4142_4344, from a big endian source across an address invariant bridge to a little endian destination.

	Big endian source bus				Little endian destination bus			
Byte lane	0	1	2	3	3	2	1	0
Address lsbs	000	001	010	011	011	010	001	000
Data	41 42 43 44				44 43 42 41			
Significance	MSB		LSB		MSB		LSB	

Figure 23-53. Address Invariant Byte Ordering—4 bytes Outbound

Note that although the significance of the bytes within the scalar have changed, the address of the individual bytes that make up the scalar have not changed. As long as software is aware that the source of the data used a big endian format, the data can be interpreted correctly.

Figure 23-54 shows data flowing the other way, from a little endian source to a big endian destination.

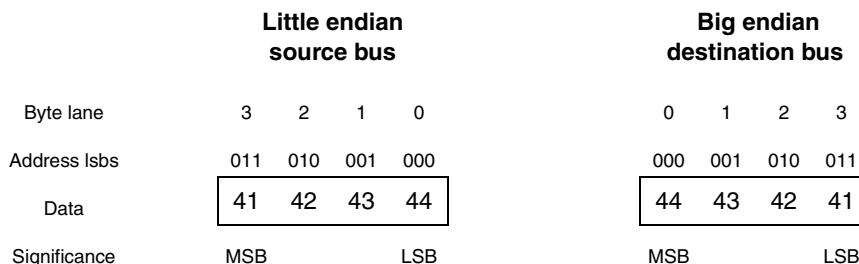


Figure 23-54. Address Invariant Byte Ordering—4 bytes Inbound

Figure 23-55 shows an outbound transfer of an 8-byte scalar, 0x5455_1617_CDCE_2728, using address invariance.

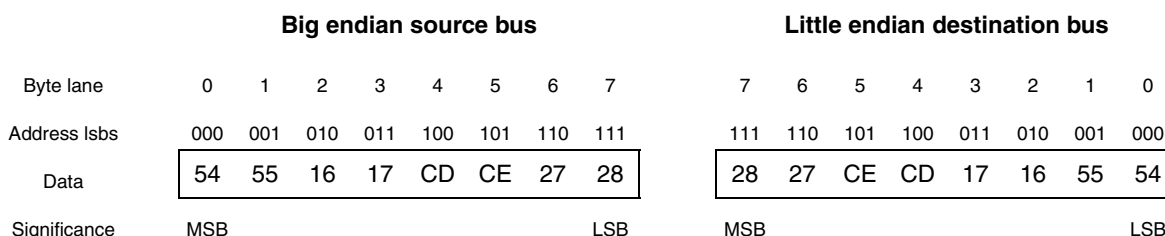


Figure 23-55. Address Invariant Byte Ordering—8 bytes Outbound

Figure 23-56 shows an inbound transfer of a 2-byte scalar, 0x5837, using address invariance.

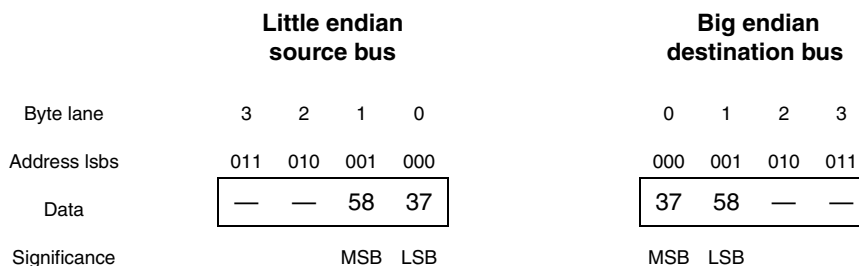


Figure 23-56. Address Invariant Byte Ordering—2 bytes Inbound

Note that in all of these examples, the original addresses of the individual bytes within the scalars (as created by the source) have been preserved.

23.4.8.1 Byte Order for Configuration Transactions

All internal memory-mapped registers in the CCSR space use big endian byte ordering. However, the PCI specification defines PCI configuration registers as little endian. All accesses to the PCI configuration port, CFG_DATA, including the those targeting the internal PCI configuration registers, use the address invariance policy as shown in Figure 23-57. Therefore, software must access CFG_DATA with little-endian formatted data—either using the **lwbrx/stwbrx** instructions or by manipulating the data before writing to and after reading from CFG_DATA.

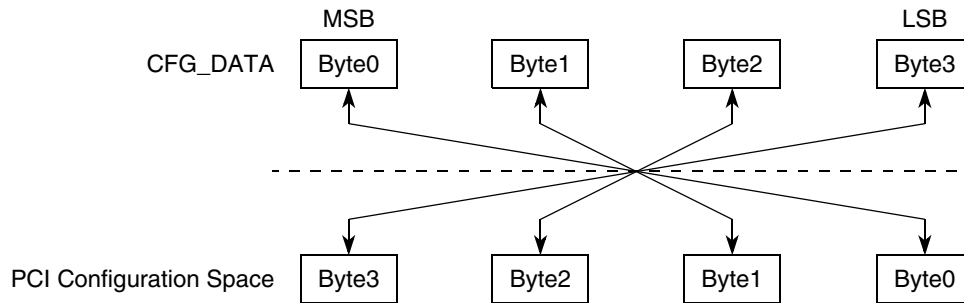


Figure 23-57. CFG_DATA Byte Ordering

23.5 Initialization/Application Information

The following sections describe initialization sequences for host and agent modes.

23.5.1 Initialization Sequence for Host Mode

The following sequence must be followed in host mode:

1. Enable PCI output clocks and select desired frequency ratios. See [Section 4.3.2, “Reset Configuration Words.”](#)
2. Wait for at least 1 ms to enable stable clocks into agent devices
3. Deactivate PCI_RESET_OUT signal for PCI. See [Table 13-3](#) for more information on PCI_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

23.5.2 Initialization Sequence for Agent Mode

The following sequence must be followed in agent mode:

1. Optionally initialize subsystem vendor ID/device ID
 - c) Initialize PCI inbound window size in PIWAR[1:3] desired window size
 - d) Unlock configuration lock in PCI function configuration register



Chapter 24

QUICC Engine Block on the MPC8309

The *QUICC Engine Block Reference Manual with Protocol Interworking* (QEIWRM) describes all the functional units of the QUICC Engine block and must be used in conjunction with this device manual and this chapter. The QEIWRM is a superset manual which includes some information not relevant to the MPC8309. QUICC Engine block in MPC8309 is a scaled down version of the QUICC Engine block in MPC8569. This chapter serves as both a general overview of the QUICC Engine block and a guide to the specific implementation of the QUICC Engine block on the MPC8309.

- [Section 24.1, “QUICC Engine Block,”](#) gives a general overview of the MPC8309 QUICC Engine architecture.
- [Section 24.2, “QUICC Engine Implementation Details for the MPC8309,”](#) lists the chapters that do apply. Implementation-specific details for some chapters follow.

24.1 QUICC Engine Block

The QUICC Engine block is a versatile communications complex that integrates several communications peripheral controllers. It provides an on-chip system design that can be used as a building block for chip integration in a variety of applications, particularly in communications and networking systems.

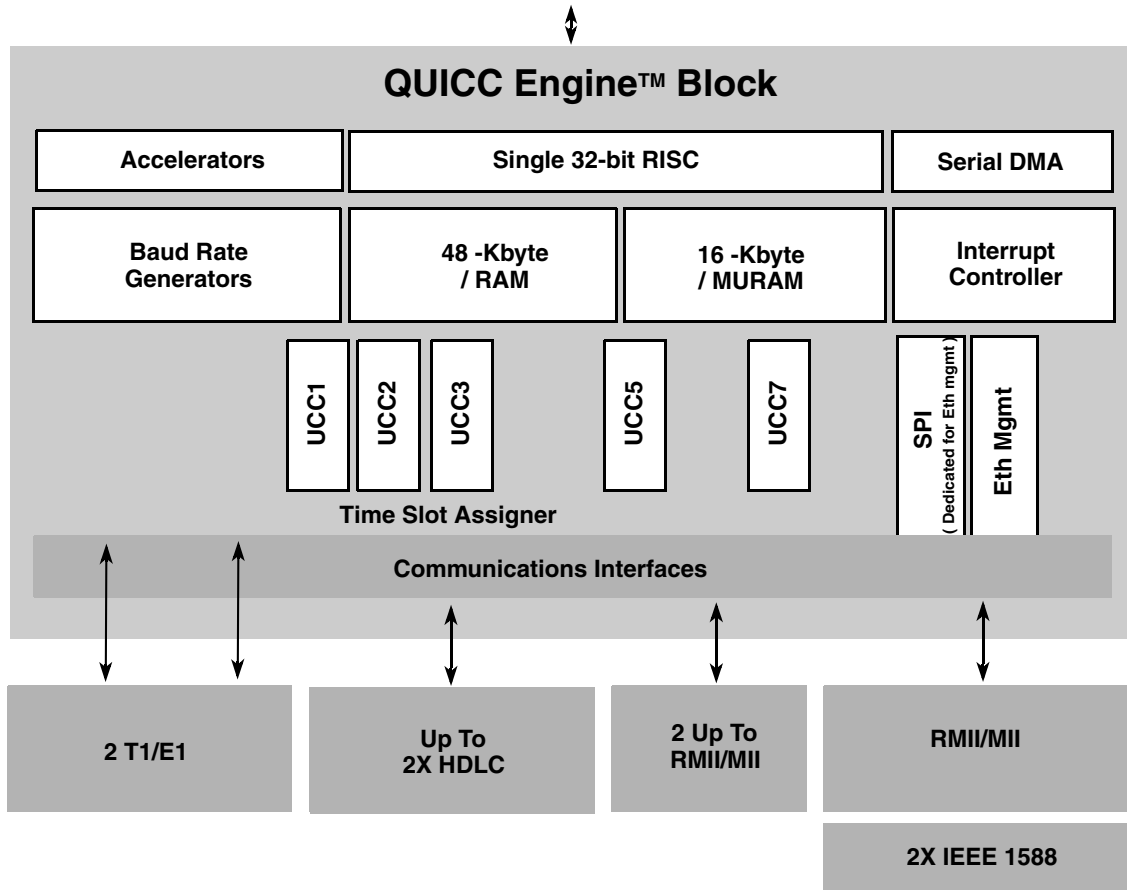
The QUICC Engine block is the next generation of the Power QUICC II CPM and maintains a high level of compatibility with it.

The QUICC Engine block contains the following communication peripherals:

- Five unified communication controllers (UCCs) with the following features:
 - 10/100 Mbps Ethernet/IEEE® Std. 802.3® through MII and RMII interfaces
 - IEEE Std. 1588™ support for MPC8309
 - Ethernet, HDLC/HDLC bus, and transparent protocols (also known as fast protocols).
 - Ethernet for the First Mile (IEEE 802.3ah 2BASE-TL and 10PASS-TS)
 - Async HDLC that are user compatible with the SCC of the CPM
 - The HDLC and transparent protocols are user compatible with the FCC of the CPM.
- UART
- BISYNC (bitrate up to 2 Mbps)
- Time slot assigner and serial interface (SI) for 2 TDMs and full duplex routing RAM of 512 entries.
- Ethernet Management Functionality by Ethernet MDC/MDIO and SPI.

The UCCs are similar to the PowerQUICC II peripherals: SCC (HDLC bus), and FCC (fast Ethernet, HDLC, and transparent).

Figure 24-1 shows the internal architecture and the interfaces provided by the QUICC Engine block on the MPC8309. A common multiuser RAM is used to store parameters for the RISC engine. The RISC has an Instruction RAM associated with it, which is loaded with the microcode image. The instruction RAM is used to optionally run additional code.



Note: Possible configuration for TDM/HDLC are- 1xTDM1xHDLC, 2TDM, 2XHDLC

Figure 24-1. QUICC Engine Block Architectural Block Diagram for the MPC8309

24.2 QUICC Engine Implementation Details for the MPC8309

Most chapters of the QEIWRM apply to the MPC8309 without modification. However, some of these chapters have application differences that are pointed out in the reference manual. They are also given in this overview section.

While using the QEIWRM for the MPC8309, use this MPC8309 implementation-specific information in general:

- e300 core
- One 32-bit RISC

- IEEE 1588 V2 standard hardware support applies
- No local bus
- Five UCCs
- No MCC
- No USB
- Only one SPI for ethernet management only
- 48K IRAM
- 16K DRAM

Table 24-1 lists the chapters from the *QUICC Engine Block Reference Manual with Protocol Interworking* (QEIWRM) and the chapters with implementation differences.

Table 24-1. QEIWRM Chapters and 8309 Implementation

Chapter Name	MPC8309 Implementation
Part I, “Introduction”	
System Interface	Applies to MPC8309 —see Section 24.2.1, “System Interface,” for MPC8309 implementation.
Configuration	Applies to MPC8309 —see Section , “,” for MPC8309 implementation.
Multiplexing and Timers	Applies to MPC8309 —see Section 24.2.3, “QUICC Engine Multiplexing and Timers,” for MPC8309 implementation.
Part II, “Unified Communication Controllers (UCCs)”	
Unified Communications Controllers (UCCs)	Applies to MPC8309
UCC for Fast Protocols	Applies to MPC8309
UCC Ethernet Controller (UEC)	Applies to MPC8309 —see Section 24.2.4, “UCC Ethernet (UEC),” for MPC8309 implementation.
IEEE Standard 1588 Assist	Applies to MPC8309 —see Section 24.2.5, “IEEE Standard 1588 Assist,” for MPC8309 implementation
UTOPIA POS Bus Controller (UPC)	Does not apply to MPC8309
ATM Controller AAL0, AAL1, and AAL5	Does not apply to MPC8309
ATM Adaptation Layer 2	Does not apply to MPC8309
UCC POS Controller (UPOS)	Does not apply to MPC8309
HDLC Controller	Applies to MPC8309
Transparent Controller	Applies to MPC8309
UCC for Slow Protocols	Applies to MPC8309
UART Mode and Asynchronous HDLC	Applies to MPC8309
Serial Peripheral Interface (SPI)	Support for MIIMCOM Mode (Ethernet PHY Management Mode) only

Table 24-1. QEIWRM Chapters and 8309 Implementation (continued)

Chapter Name	MPC8309 Implementation
Universal Serial Bus Controller	Does not apply to MPC8309
BISYNC Mode	Applies to MPC8309
Part III, “Time Division Multiplex Support (TDM)”	
Serial Interface with Time-Slot Assigner	Applies to MPC8309
Multi-Channel Controller (MCC)	Does not apply to MPC8309
Multi-Channel Controller on UCC (UMCC)	Applies to MPC8309
QUICC Multi-Channel Controller (QMC)	Does not apply to MPC8309
Point-to-Point Protocol (PPP)	Does not apply to MPC8309
Serial ATM Microcode	Does not apply to MPC8309
Inverse Multiplexing for ATM (IMA)	Does not apply to MPC8309
ATM AAL1 Circuit Emulation Service	Does not apply to MPC8309
Part IV “Multiprotocol Interworking”	
Interworking Introduction	Does not apply to MPC8309
Frame Parse and Lookup	Does not apply to MPC8309
Protocol Interworking Programming Model	Does not apply to MPC8309
Virtual Port	Does not apply to MPC8309
IP Reassembly	Does not apply to MPC8309
IPv4/UDP Header Compression	Does not apply to MPC8309
IPsec Microcode Package	Does not apply to MPC8309
Part IV, “Switching Functionality”	
Enhanced MSP Microcode	Does not apply to MPC8309
L2 Ethernet Switch	Does not apply to MPC8309

The following subsections include MPC8309-specific details for the given chapters of the QEIWRM.

24.2.1 System Interface

The information in this MPC8309-specific “System Interface” subsection applies to “System Interface” chapter of the QEIWRM.

24.2.1.1 System Interface—Serial DMA

In the “Serial DMA” subsection, the following information applies for the MPC8309. The QUICC Engine module has one physical serial DMA (SDMA) channel. On the MPC8309, one channel interfaces with coherent system bus, the other channel interfaces with the secondary bus.

24.2.1.2 System Interface—Data Paths

In the “Data Paths” subsection, use this information:

Figure 24-2 is a simplified MPC8309 block diagram that shows the data paths. They can be configured with one DDR bus (16 bit data).

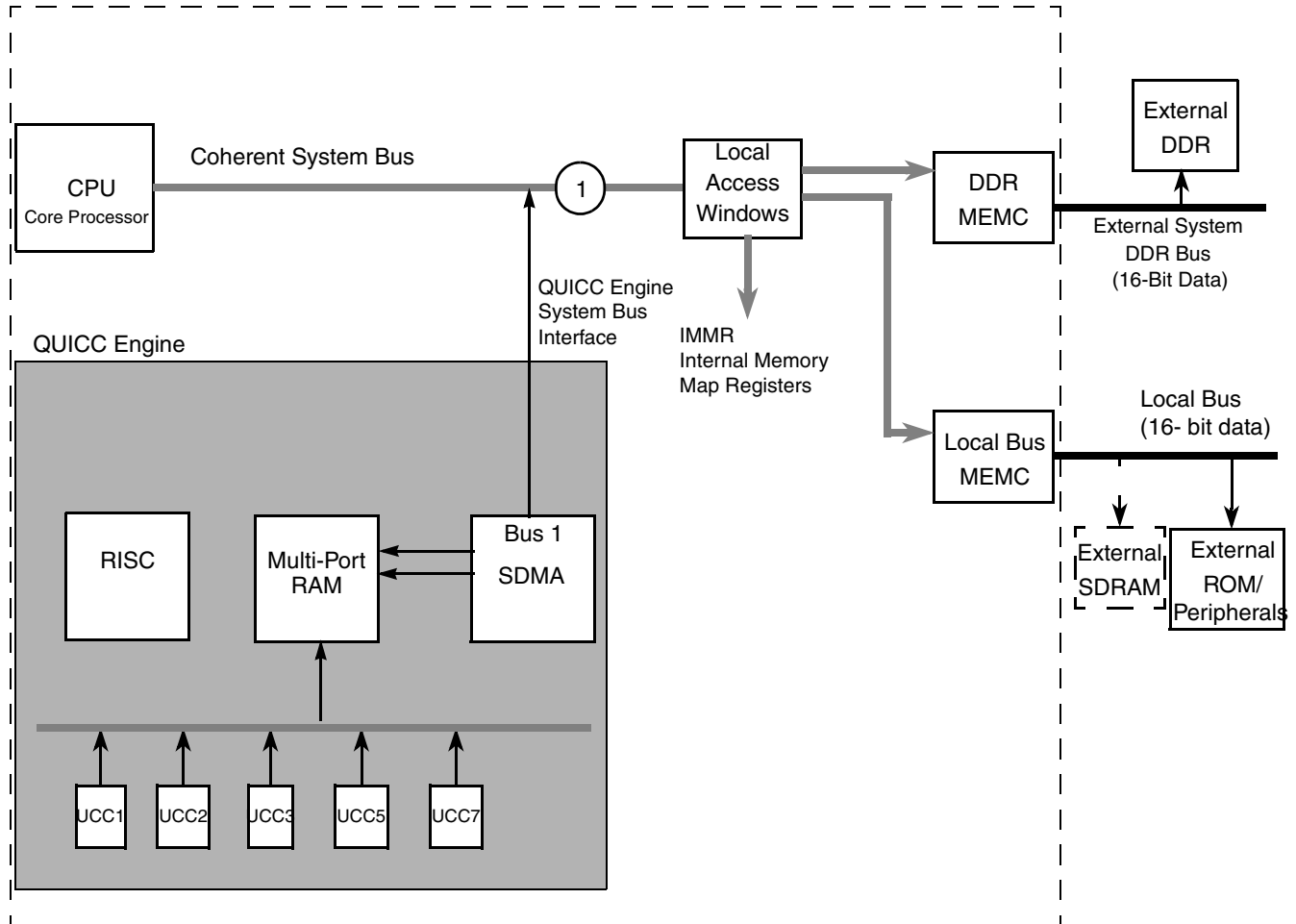


Figure 24-2. Data Paths

24.2.1.3 System Interface—SDMA and Bus Error

In the “SDMA and Bus Error” subsection, use the following information: Under 3b. “On the MPC8309, the device is reset by asserting the hard reset signal, HRESET, (the reset command to the QUICC Engine Command Register is not sufficient).”

24.2.1.4 System Interface—SDMA and Reset

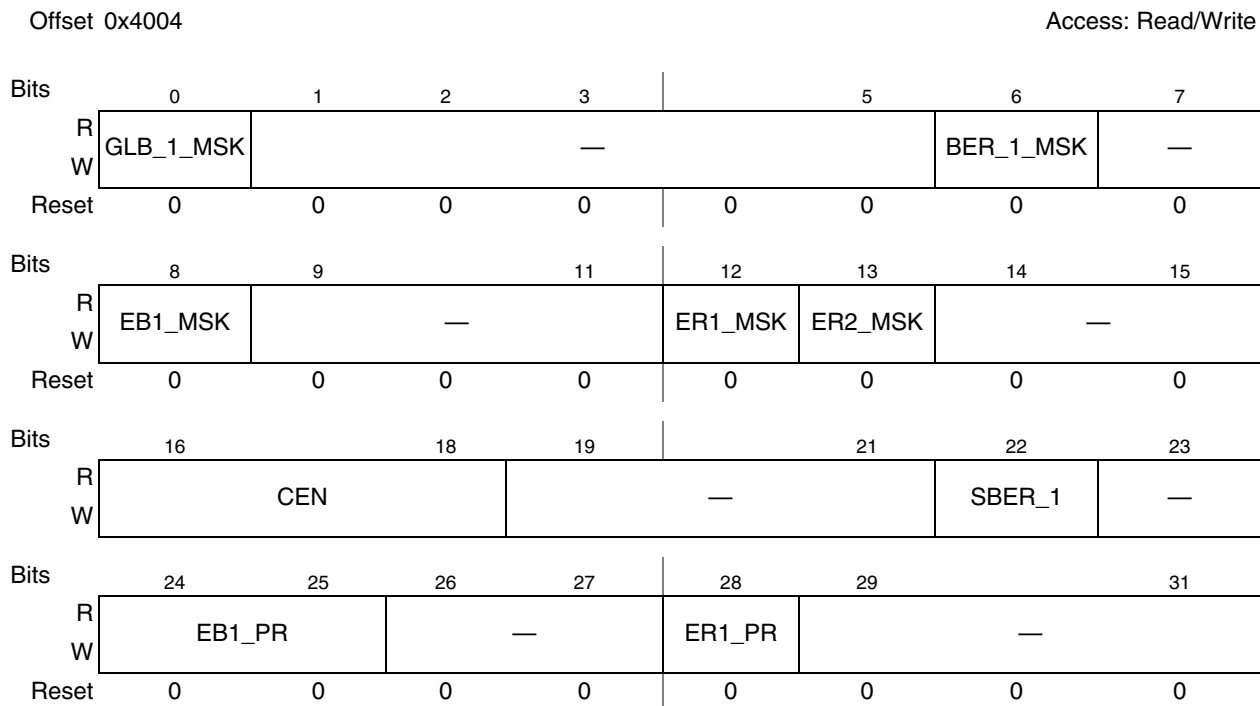
In the “SDMA and Reset” subsection, use the following information: During system reset (on the MPC8309) all SDMA FIFOs are flushed and all outstanding transactions are stopped.

Table 24-2. SDRS Field Descriptions (continued)

Bits	Name	Description
7	BER_2	Bus 2 error event. Used to indicate that the Serial DMA channel on bus 2 terminated with an error during a read or write transaction. This bit is cleared by writing '1'. Writing '0' has no effect. The Serial DMA bus 2 error address is read from the SDTA2 register. The communication channel number is read from the SDTM2 register. 0 No bus error event occurred on bus 2 1 A bus error event occurred on bus 2
8–31	—	Reserved, should be cleared.

24.2.1.6.2 Serial DMA Mode Register (SDMR)

SDMR, shown in [Figure 24-4](#), enables the user to mask the Serial DMA interrupts and emergency mode and to set them manually. For bits 6,7 if an SDMR bit is set, the corresponding interrupt in SDRS is enabled. If the bit is cleared, the corresponding interrupt in SDRS is masked.


Figure 24-4. Serial DMA Mode Register (SDMR)

[Table 24-3](#) describes the SDMR fields.

Table 24-3. SDMR Field Descriptions

Bits	Name	Description
0	GLB_1_MSK	Mask global mode on bus 0 Mask global mode on bus 1 Enable global mode on bus
1–5	—	Reserved, should be cleared.

Table 24-3. SDMR Field Descriptions (continued)

Bits	Name	Description
6	BER_1_MSK	Mask bus error events. 0 Mask bus error events 1 Enable bus error events
7	—	Reserved, should be cleared.
8	EB1_MSK	Mask emergency on external bus 0 Mask emergency towards External bus 1 Enable emergency towards External bus
9–11	—	Reserved, should be cleared.
12	ER1_MSK	Mask emergency on Multi-user RAM port 1. 0 Mask Multi-user RAM port 1 emergency 1 Enable Multi-user RAM port 1 emergency
13	ER2_MSK	Mask emergency on Multi-user RAM port 2. 0 Mask Multi-user RAM port 2 emergency 1 Enable Multi-user RAM port 2 emergency
14–15	—	Reserved, should be cleared.
16–18	CEN	SDMA Temporary Buffer Size: 000 512 Bytes 001 1 Kbytes 010 1.5 Kbytes 011 2 KBytes 100 2.5 Kbytes 101 3 KBytes
19–21	—	Reserved, should be cleared.
22	SBER_1	Stop at bus error event on bus 0 Continue DMA transactions regularly, even if a bus error occurred on bus 1 Stop the DMA transactions (after ending all current open dma transactions) if a bus error occurred on bus
23	—	Reserved, should be cleared.
24–25	EB1_PR	Set priority on bus (Used when the SDMA is not in emergency state or when emergency requests are masked by EB1_MSK. When the SDMA is in emergency state, highest priority is used). 00 Level 0 (Lowest) 01 Level 1 10 Level 2 11 Level 3 (Highest)
26–27	—	Reserved, should be cleared.
28	ER1_PR	Set priority on Multi-user RAM port 1 (Used when the SDMA is not in emergency state or when emergency requests are masked by ER1_MSK. When the SDMA is in emergency state, highest priority is used). 0 Low priority. 1 High priority.

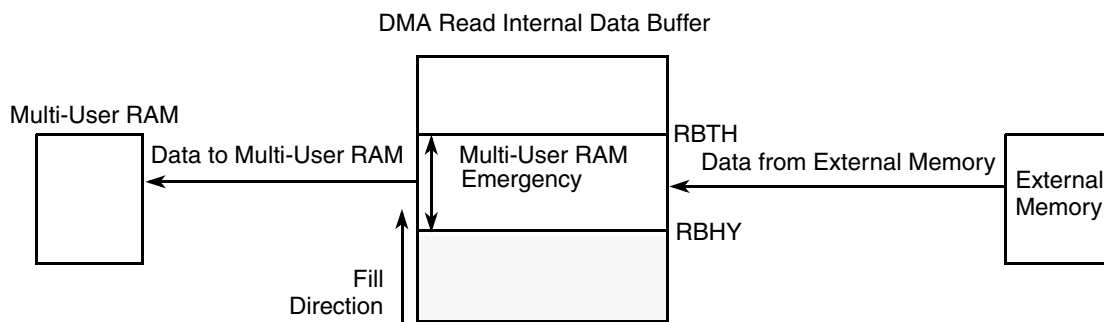
Table 24-3. SDMR Field Descriptions (continued)

Bits	Name	Description
29	ER2_PR	Set priority on Multi-user RAM port 2 (Used when the SDMA is not in emergency state or when emergency requests are masked by ER2_MSK. When the SDMA is in emergency state, highest priority is used.). 0 Low priority. 1 High priority.
30–31	—	Reserved, should be cleared.

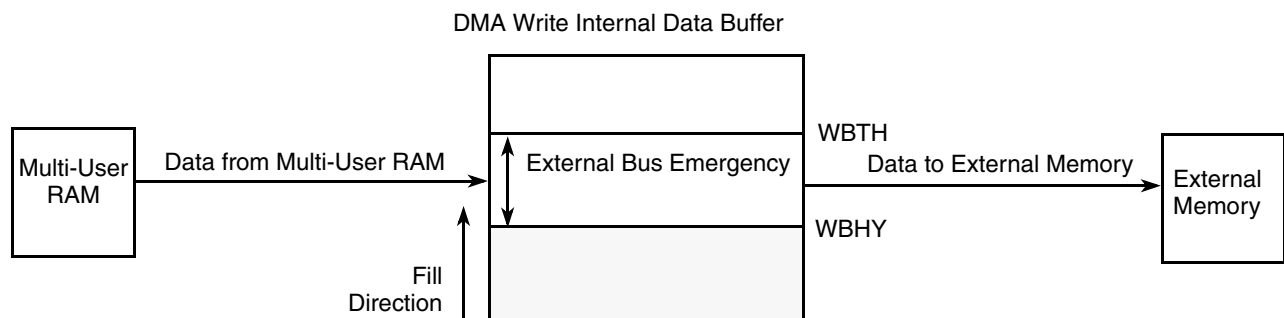
24.2.1.6.3 Serial DMA Threshold Registers (SDTR)

SDTR, shown in [Figure 24-8](#), with a combination of SDHY defines the high priority levels towards Multi-user RAM and External buses. High priority is activated when reaching the threshold value. The high priority will be held until reaching the hysteresis value.

When the internal DMA read data buffer (buffer for DMA read commands) reaches its threshold value (RBTH), a high priority indicator towards Multi-user RAM port will be asserted, and will be held asserted until the data buffer reaches its hysteresis value (RBHY).


Figure 24-5. DMA Read Data Path

When the internal DMA write data buffer (buffer for DMA write commands) reaches its threshold value (WBTH), a high priority indicator towards the External bus will be asserted, and will be held asserted until the data buffer reaches the hysteresis value (WBHY).


Figure 24-6. DMA Write Data Path

The DMA contains a command queue for each External bus. When the DMA command queue reaches its threshold (CQTH), a high priority indicator towards the Multi-user RAM port and External bus will be asserted, and will be held asserted until the command queue reaches the hysteresis value (CQHY).

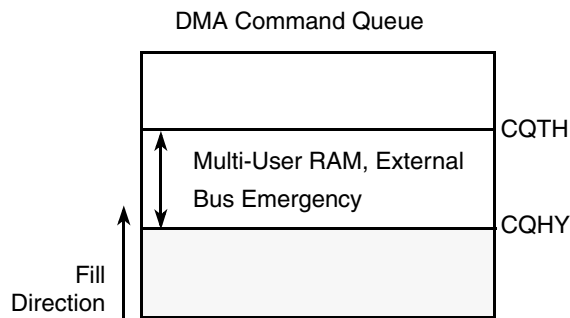


Figure 24-7. DMA Command Queue

Offset SDTR: 0x4008

Access: Read/Write

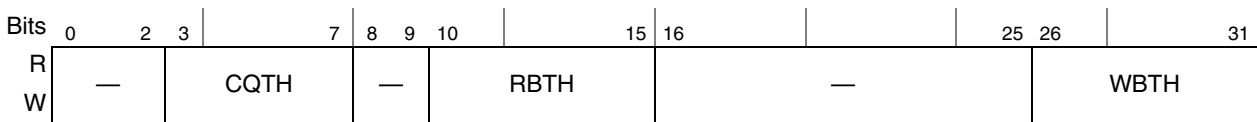


Figure 24-8. Serial DMA Threshold Register (SDTR)

Table 24-4 describes the SDTRx fields.

Table 24-4. SDTR Field Descriptions

Bits	Name	Description
0–2	—	Reserved, should be cleared.
3–7	CQTH	Command queue threshold. When the command queue reaches its threshold value, a high priority indication towards Multi-user RAM port and external address bus will be asserted
8–9	—	Reserved, should be cleared.
10–15	RBTH	Read internal data buffer threshold. When the Read internal data buffer reaches its threshold value, a high priority indication towards Multi-user RAM port will be asserted
16–25	—	Reserved, should be cleared.
26–31	WBTH	Write internal data buffer threshold. When the Write internal data buffer reaches its threshold value, a high priority indication towards the External bus will be asserted

24.2.1.6.4 Serial DMA Hysteresis Registers (SDHY)

SDHY1 and SDHY2, shown in Figure 24-9, with a combination of SDTR defines the high priority levels towards Multi-user RAM and External buses. High priority is activated when the parameter (internal data buffer or command queue in this case) reaches the threshold values, found in the SDTR registers. The high priority indicator will be held until the parameter reaches the hysteresis value, found in the SDHY

registers. For a full explanation, see [Section 24.2.1.6.3, “Serial DMA Threshold Registers \(SDTR\),” on page 24-9.](#)

Offset SDHY1: 0x4010

Access: Read/Write

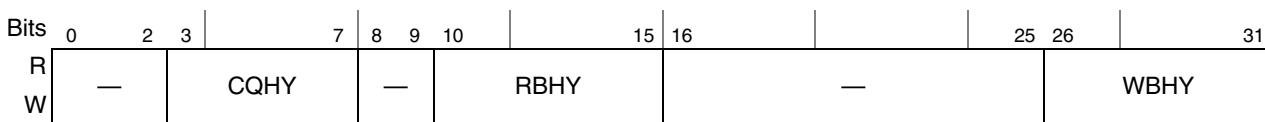


Figure 24-9. Serial DMA Hysteresis Register (SDHY)

Table 24-5 describes the SDHY fields.

Table 24-5. SDHYx Field Descriptions

Bits	Name	Description
0–2	—	Reserved, should be cleared.
3–7	CQHY	Command queue hysteresis. When the command queue reaches its threshold value (CQTH), a high priority indicator towards Multi-user RAM port and external address bus will be asserted. It will be held until reaching the hysteresis value stored in these bits (CQHY).
8–9	—	Reserved, should be cleared.
10–15	RBHY	Read internal data buffer hysteresis. When the Read internal data buffer reaches its threshold value (RBTH), a high priority indicator towards Multi-user RAM port will be asserted. It will be held until reaching the hysteresis value stored in these bits (RBHY).
16–25	—	Reserved, should be cleared.
26–31	WBHY	Write internal data buffer hysteresis. When the Write internal data buffer reaches its threshold value (WBTH), a high priority indicator towards the External bus will be asserted. It will be held until reaching the hysteresis value stored in these bits (WBHY).

24.2.1.6.5 Serial DMA Transfer Address Registers (SDTA)

SDTA, shown in [Figure 24-10](#), are read only registers, that hold the address accessed during the current bus transaction. In case of bus error, the address is locked in the register until the corresponding status bit in the SDSR register is reset.

Offset SDTA1: 0x4018

Access: w1c

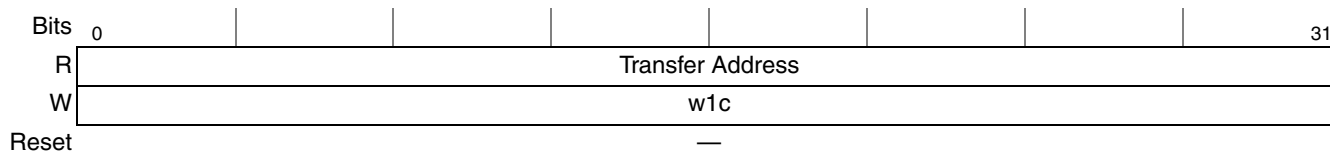


Figure 24-10. Serial DMA Transfer Address Register (SDTA)

Table 24-6 describes the SDTAX fields.

Table 24-6. SDTAX Field Descriptions

Bits	Name	Description
0–31	Transfer Address	Address accessed during current bus transaction. The value is updated at every transfer end. The value is locked in case of bus error and released when the SDSR[BER_x] bit is reset by the user (writing '1' to it).

24.2.1.6.6 Serial DMA Transfer Communication Channel Number Registers (SDTM)

SDTM, shown in Figure 24-11, are read only registers that hold the serial number of the peripheral that was served during the current bus transaction. In case of bus error, the MSNUM value is locked in the register until the corresponding status bit in the SDSR register is reset.

Offset SDTM1: 0x4020

Access: w1c

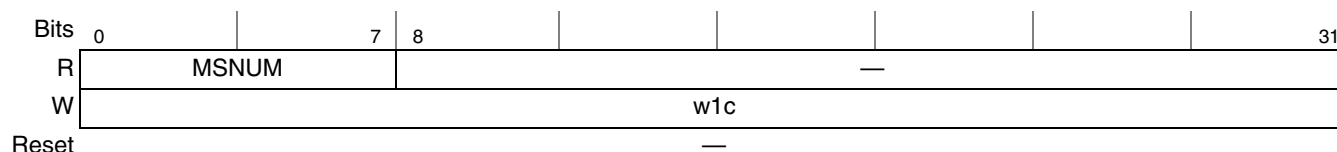


Figure 24-11. Serial DMA Transfer MSNUM Register (SDTM)

Table 24-7 describes the SDTM fields.

Table 24-7. SDTMx Field Descriptions

Bits	Name	Description
0–7	MSNUM	MSNUM served during current bus transaction. The value is updated at every transfer end. The value is locked in case of bus error and released when the SDSR[BER_x] bit is reset by the user (writing '1' to it).
8–31	—	Reserved, should be cleared.

24.2.1.6.7 Serial DMA Temporary Buffer Base in Multi-User RAM Value (SDEBCR)

SDEBCR, shown in Figure 24-12, holds the Temporary Buffer base address in multi-user RAM.

Offset 0x4044

Access: Read/Write



Figure 24-12. Serial DMA Temporary Buffer Base in Multi-User RAM Value (SDEBCR)

Table 24-8 describes the SDEBCR fields.

Table 24-8. SDEBCR Field Descriptions

Bits	Name	Description
0–6	—	Reserved, should be cleared.
7–31	BA	Temporary Buffer base address in Multi-user RAM. The address must be 4KB aligned.

24.2.1.7 Interrupt Controller

The peripherals of the QUICC Engine module (UCCs, USB, MCC, SDMA, Timers, and SPIs where applicable) are the main interrupt sources. It is possible to program a priority hierarchy, and assign a unique identifier (vector) to the interrupts from these sources. The QUICC Engine module offers two priority schemes—a grouped hierarchy, and a spread hierarchy.

24.2.1.7.1 Interrupt Configuration

Figure 24-13 shows the QUICC Engine interrupt structure. The interrupt controller receives interrupts from various internal sources within the QUICC Engine module.

Figure 24-13. QUICC Engine Module Interrupt Structure

The interrupt controller allows masking of each interrupt source. In addition, multiple events within a QUICC Engine module peripheral event are also maskable. The Interrupt controller can be programmed to split the interrupts between two interrupt outputs: QUICC Engine High and QUICC Engine Low.

All interrupt sources are prioritized and bits are set in the interrupt pending register (CIPNR). On the QUICC Engine module, the prioritization of the interrupt sources can be programmed in two areas:

The relative priority of the UCCs, USB, MCC, and SPIs. See [Section 24.2.1.7.3, “UCC Relative Priority,”](#) for more information.

- Assigning highest priority to one interrupt source. See [Section 24.2.1.7.4, “Highest Priority Interrupt,”](#) for more information.

Each interrupt output (QUICC Engine High, QUICC Engine Low) has its corresponding interrupt vector register. The QUICC Engine interrupt vector register (CIVEC) and QUICC Engine High interrupt vector register (CHIVEC) are updated with a 6-bit vector corresponding to the peripheral with the current highest priority within the group assigned to each interrupt output.

24.2.1.7.2 Interrupt Source Priorities

The interrupt controller has interrupt sources that assert two interrupt requests outputs (QUICC Engine High and QUICC Engine Low). The user is able to allocate some interrupt sources to the QUICC Engine High interrupt output (see [Section 24.2.1.8.2, “QUICC Engine System Interrupt Control Register \(CICNR\)”](#)). All the sources can be allocated to the QUICC Engine Low interrupt output. [Table 24-9](#) shows prioritization of all interrupt sources. As described in the following sections, flexibility exists in the relative ordering of the interrupts, but, in general, relative priorities are not shown. Each interrupt priority number corresponds to each table entry.

Note that the group and spread options, shown with XCC and YCC entries in [Table 24-9](#), are described in [Section 24.2.1.7.3, “UCC Relative Priority.”](#)

Table 24-9. Interrupt Source Priority Levels

Priority Level	Interrupt Source Description	Multiple Events
1	Highest	—
2	MIXA1 (Grouped/Spread)	Yes
3	MIXA2 (Grouped)	Yes
4	MIXA3 (Grouped)	Yes
5	MIXA4 (Grouped)	Yes
6	RTB1 (Spread)	Yes
7	XCC1 (Grouped)	Yes
8	XCC2 (Grouped)	Yes
9	XCC3 (Grouped)	Yes
10	XCC4 (Grouped)	Yes
11	MIXA2 (Spread)	Yes
12	XCC5 (Grouped)	Yes
13	XCC6 (Grouped)	Yes
14	XCC7 (Grouped)	Yes
15	XCC8 (Grouped)	Yes
16	RTB1 (Grouped)	Yes
17	RTB2 (Grouped)	Yes
18	RTB3 (Grouped)	Yes
19	RTB4 (Grouped)	Yes
20	RTB2 (Spread)	Yes
21	YCC1 (Grouped)	Yes
22	YCC2 (Grouped)	Yes
23	YCC3 (Grouped)	Yes
24	YCC4 (Grouped)	Yes
25	MIXA3 (Spread)	Yes
26	YCC5 (Grouped)	Yes
27	YCC6 (Grouped)	Yes
28	YCC7 (Grouped)	Yes
29	YCC8 (Grouped)	Yes
30	MIXA5 (Grouped)	Yes
31	MIXA6 (Grouped)	Yes

Table 24-9. Interrupt Source Priority Levels (continued)

Priority Level	Interrupt Source Description	Multiple Events
32	MIXA7 (Grouped)	Yes
33	MIXA8 (Grouped)	Yes
34	RTB3 (Spread)	Yes
35	WCC1 (Grouped)	Yes
36	WCC2 (Grouped)	Yes
37	WCC3 (Grouped)	Yes
38	WCC4 (Grouped)	Yes
39	MIXA4 (Spread)	Yes
40	WCC5 (Grouped)	Yes
41	WCC6 (Grouped)	Yes
42	WCC7 (Grouped)	Yes
43	WCC8 (Grouped)	Yes
44	RTB5 (Grouped)	Yes
45	RTB6 (Grouped)	Yes
46	RTB7 (Grouped)	Yes
47	RTB8 (Grouped)	Yes
48	RTB4 (Spread)	Yes
49	ZCC1 (Grouped)	Yes
50	ZCC2 (Grouped)	Yes
51	ZCC3 (Grouped)	Yes
52	ZCC4 (Grouped)	Yes
53	MIXA5 (Spread)	Yes
54	ZCC5 (Grouped)	Yes
55	ZCC6 (Grouped)	Yes
56	ZCC7 (Grouped)	Yes
57	ZCC8 (Grouped)	Yes
58	MIXA5 (Spread)	Yes
59	Reserved	—
60	XCC1 (Spread)	Yes
61	YCC1 (Spread)	Yes
62	Reserved	—
63	WCC1 (Spread)	Yes
64	ZCC1 (Spread)	Yes

Table 24-9. Interrupt Source Priority Levels (continued)

Priority Level	Interrupt Source Description	Multiple Events
65–66	Reserved	—
67	MIXA6 (Spread)	Yes
68	Reserved	—
69	XCC2 (Spread)	Yes
70	YCC2 (Spread)	Yes
71	Reserved	—
72	WCC2 (Spread)	Yes
73	ZCC2 (Spread)	Yes
74–75	Reserved	—
76	RTB6 (Spread)	Yes
77	Reserved	—
78	XCC3 (Spread)	Yes
79	YCC3 (Spread)	Yes
80	Reserved	—
81	WCC3 (Spread)	Yes
82	ZCC3 (Spread)	Yes
83–84	Reserved	—
85	MIXA7 (Spread)	Yes
86	Reserved	—
87	XCC4 (Spread)	Yes
88	YCC4 (Spread)	Yes
89	Reserved	—
90	WCC4 (Spread)	Yes
91	ZCC4 (Spread)	Yes
92–93	Reserved	—
94	RTB7 (Spread)	Yes
95	Reserved	—
96	XCC5 (Spread)	Yes
97	YCC5 (Spread)	Yes
98	Reserved	—
99	WCC5 (Spread)	Yes
100	ZCC5 (Spread)	Yes
101–102	Reserved	—

Table 24-9. Interrupt Source Priority Levels (continued)

Priority Level	Interrupt Source Description	Multiple Events
103	MIXA8 (Spread)	Yes
104	Reserved	—
105	XCC6 (Spread)	Yes
106	YCC6 (Spread)	Yes
107	Reserved	—
108	WCC6 (Spread)	Yes
109	ZCC6 (Spread)	Yes
110–111	Reserved	—
112	RTB8 (Spread)	Yes
113	Reserved	—
114	XCC7 (Spread)	Yes
115	YCC7 (Spread)	Yes
116	Reserved	—
117	WCC7 (Spread)	Yes
118	ZCC7 (Spread)	Yes
119–121	Reserved	—
122	XCC8 (Spread)	Yes
123	YCC8 (Spread)	Yes
124	Reserved	—
125	WCC8 (Spread)	Yes
126	ZCC8 (spread)	Yes

There are two ways to program the flexibility of the QUICC Engine module interrupt priorities:

- The UCC relative priority option. See [Section 24.2.1.7.3, “UCC Relative Priority,”](#) for more information.
- The highest priority option. See [Section 24.2.1.7.4, “Highest Priority Interrupt,”](#) for more information.

24.2.1.7.3 UCC Relative Priority

The relative priority between the UCCs is programmable and can be changed dynamically. There is no entry for UCC1-4 and UCC5-8 in [Table 24-9](#), but rather there are entries for XCC1–XCC8 and YCC1–YCC8. UCC5-8 and UCC1-4 can be mapped to any YCC location and any XCC location, respectively. The UCC priorities are programmed in the QUICC Engine interrupt priority registers (CIPXCC and CIPYCC) and can be changed dynamically to implement a rotating priority.

In addition, the grouping of the locations of the XCC and YCC entries has the following two options:

- **Group**—In the group scheme, all UCCs are grouped together at the top of the priority table, ahead of most other QUICC Engine interrupt sources. This scheme is ideal for applications where all UCCs 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 that the other sources can have lower interrupt latencies. This scheme is also programmed in the CICR but cannot be changed dynamically.

NOTE: On Backward Compatibility

The allocation of the interrupts allows for backward compatibility with the 82xx/85xx family of devices that are listed in [Table 24-10](#).

Table 24-10. Mapping of FCCs and SCCs to UCCs for 82xx/85xx Compatibility

82xx Device	QUICC Engine Device
FCC1	UCC1
FCC2	UCC2
FCC3	UCC3
SCC1	UCC5
SCC3	UCC7

24.2.1.7.4 Highest Priority Interrupt

In addition to the UCC relative priority option, CICR[HP] can be used to specify one interrupt source based on its highest priority. This interrupt remains within the same interrupt level as the other interrupt controller interrupts, but it is serviced before any other interrupt in the table.

CICR[HP] can be updated dynamically to allow the user to change a normally low priority source into a high priority-source for a certain period.

24.2.1.7.5 Masking Interrupt Sources

By programming the system interrupt mask register (CIMR), the user can mask interrupt requests to the core. Each CIMR bit corresponds to an interrupt source. The corresponding CIMR bit is set to enable an interrupt. When a masked interrupt source has a pending interrupt request, the corresponding CIPNR bit is set, even though the interrupt is not sent to the system interrupt controller. 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 block. [Table 24-9](#) shows the interrupt sources that have multiple interrupting events. [Figure 24-14](#) shows an example on how the masking occurs, using a UCC.

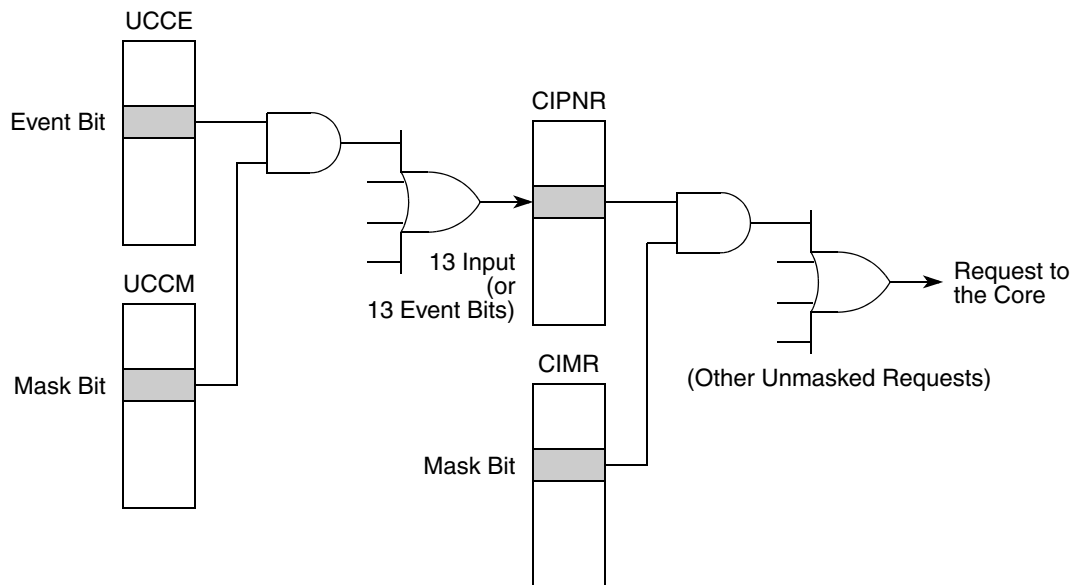


Figure 24-14. Interrupt Request Masking

24.2.1.7.6 Interrupt Vector Generation and Calculation

Pending unmasked interrupts are presented to the core processor according to priority. The interrupt vector that allows the core processor to locate the interrupt service routine is made available to the core processor by reading CIVEC for QUICC Engine Low interrupt output and CHIVEC for QUICC Engine High interrupt output. The interrupt controller passes an interrupt vector according to the highest-priority, unmasked, pending interrupt. [Table 24-11](#) lists encoding for the six low-order bits of the interrupt vector.

Table 24-11. Encoding the Interrupt Vector

Interrupt Number ¹	Interrupt Source Description	Interrupt Vector
0	Error (No interrupt)	0b00_0000
1	SPI2	0b00_0001
2	SPI1	0b00_0010
3	RTT	0b00_0011
4	Reserved	0b00_0100
5	Reserved	0b00_0101
6	Reserved	0b00_0110
7	Reserved	0b00_0111
8	Reserved	0b00_1000
9	Reserved	0b00_1001
10	SDMA	0b00_1010
11	USB	0b00_1011
12	Timer1	0b00_1100

Table 24-11. Encoding the Interrupt Vector (continued)

Interrupt Number ¹	Interrupt Source Description	Interrupt Vector
13	Timer2	0b00_1101
14	Timer3	0b00_1110
15	Timer4	0b00_1111
16	Reserved	0b01_0000
17	PTP1 ² ,	0b01_0001
18	Reserved	0b01_0010
19	Reserved	0b01_0011
20	VT	0b01_0100
21	RTC ²	0b01_0101
22–24	Reserved	0b01_0110–0b01_1000
25	EXT1	0b01_1001
26	EXT2	0b01_1010
27	EXT3	0b01_1011
28	EXT4	0b01_1100
29–31	Reserved	0b01_1101–0b01_1111
32	UCC1	0b10_0000
33	UCC2	0b10_0001
34	UCC3	0b10_0010
35	Reserved	0b10_0011
36	MCC1	0b10_0100
37	Reserved	0b10_0101
38	Reserved	0b10_0110
39	Reserved	0b10_0111
40	UCC5	0b10_1000
41	Reserved	0b10_1001
42	UCC7	0b10_1010
43	Reserved	0b10_1011
44		0b10_1100
45	Reserved	0b10_1101
46	Reserved	0b10_1110
47	Reserved	0b10_1111
48	Reserved	0b11_0000
49	Reserved	0b11_0001
50–63	Reserved	0b11_0010–0b11_1111

¹ The interrupt number is used in the HP field of the CICR register.

² For IEEE 1588 support.

Note that the interrupt vector table differs from the interrupt priority table in the following two ways:

- The vectors are fixed; they are not affected by the group mode, spread mode, or the relative priority order of the peripherals.
- An error vector exists as the first entry in [Table 24-11](#). The error vector is issued when no interrupt is requesting service.

24.2.1.8 Programming Model

The interrupt controller registers are described in the following sections:

- [Section 24.2.1.8.1, “QUICC Engine System Interrupt Configuration Register \(CICR\)”](#)
- [Section 24.2.1.8.2, “QUICC Engine System Interrupt Control Register \(CICNR\)”](#)
- [Section 24.2.1.8.3, “QUICC Engine System RISC Interrupts Control Register \(CRICR\)”](#)
- [Section 24.2.1.8.4, “QUICC Engine System Interrupt Priority Register for WCC Peripherals \(CIPWCC\)”](#)
- [Section 24.2.1.8.5, “QUICC Engine System Interrupt Priority Register for XCC Peripherals \(CIPXCC\)”](#)
- [Section 24.2.1.8.6, “QUICC Engine System Interrupt Priority Register for YCC Peripherals \(CIPYCC\)”](#)
- [Section 24.2.1.8.7, “QUICC Engine System Interrupt Priority Register for ZCC Peripherals \(CIPZCC\)”](#)
- [Section 24.2.1.8.8, “QUICC Engine System Interrupt Priority Register for RISC Tasks A \(CIPRTA\)”](#)
- [Section 24.2.1.8.9, “QUICC Engine System Interrupt Priority Register for RISC Tasks B \(CIPRTB\)”](#)
- [Section 24.2.1.8.10, “QUICC Engine System Interrupt Pending Register \(CIPNR\)”](#)
- [Section 24.2.1.8.11, “QUICC Engine System Interrupt Mask Register \(CIMR\)”](#)
- [Section 24.2.1.8.12, “QUICC Engine RISC Interrupt Pending Register \(CRIPNR\)”](#)
- [Section 24.2.1.8.13, “QUICC Engine RISC Interrupt Mask Register \(CRIMR\)”](#)
- [Section 24.2.1.8.14, “QUICC Engine System Interrupt Vector Register \(CIVEC\)”](#)
- [Section 24.2.1.8.15, “QUICC Engine High System Interrupt Vector Register \(CHIVEC\)”](#)

24.2.1.8.1 QUICC Engine System Interrupt Configuration Register (CICR)

CICR, shown in [Figure 24-15](#), defines the highest priority interrupt and whether interrupts are grouped or spread according to the priority table, [Table 24-9](#).

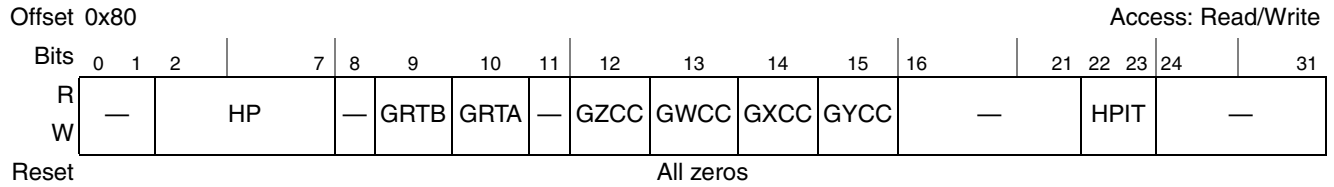


Figure 24-15. QUICC Engine System Interrupt Configuration Register (CICR)

The CICR register bits are described in [Table 24-12](#).

Table 24-12. CICR Field Descriptions

Bits	Name	Description
0–1	—	Reserved, must be cleared.
2–7	HP	Highest priority. These bits specify the 6-bit interrupt number of the single interrupt controller interrupt source that is advanced in the table based on the highest priority. HP can be modified dynamically. To retain the original priority, HP is programmed to the interrupt number assigned to MIXA1.
8	—	Reserved, should be cleared.
9	GRTB	RTB (RISC Tasks B interrupts) Priority Scheme. GRTB selects the relative RTB priority scheme and cannot be changed dynamically. 0 Grouped. The RTBs are grouped by priority at the top of the table. 1 Spread. The RTBs are spread in the table according to priority.
10	GRTA	RTA (RISC Tasks A interrupts) Priority Scheme. GRTA selects the relative RTA priority scheme and cannot be changed dynamically. 0 Grouped. The RTAs are grouped by priority at the top of the table. 1 Spread. The RTAs are spread in the table according to priority.
11	—	Reserved, should be cleared.
12	GZCC	ZCC Priority Scheme. GZCC selects the relative ZCC priority scheme and cannot be changed dynamically. 0 Grouped. The ZCCs are grouped by priority at the top of the table. 1 Spread. The ZCCs are spread in the table according to priority.
13	GWCC	WCC Priority Scheme. GWCC selects the relative WCC priority scheme and cannot be changed dynamically. 0 Grouped. The WCCs are grouped by priority at the top of the table. 1 Spread. The WCCs are spread in the table according to priority.
14	GXCC	XCC Priority Scheme. GXCC selects the relative Group1 priority scheme and cannot be changed dynamically. 0 Grouped. The XCCs are grouped by priority at the top of the table. 1 Spread. The XCCs are spread in the table according to priority.
15	GYCC	YCC Priority scheme. GYCC selects the relative YCC priority scheme and cannot be changed dynamically. 0 Grouped. The YCCs are grouped by priority at the top of the table. 1 Spread. The YCCs are spread in the table according to priority.
16–21	—	Reserved, should be cleared.
22–23	HPIT	Highest Priority Interrupt position. Defines the interrupt output that is asserted by the highest priority interrupt. 00 QUICC Engine Low is asserted when the highest priority interrupt occurs. 01 Reserved. 10 QUICC Engine High is asserted when the highest priority interrupt occurs. 11 Reserved.
24–31	—	Reserved, should be cleared.

24.2.1.8.2 QUICC Engine System Interrupt Control Register (CICNR)

CICNR, shown in [Figure 24-16](#), defines the output interrupt type (QUICC Engine High or QUICC Engine Low) in the XCC1, XCC2, YCC1, YCC2, WCC1, WCC2, ZCC1, and ZCC2 priority positions. All other priority positions will assert QUICC Engine Low signal unless they are selected as highest priority interrupt (see bit 22 in [Section 24.2.1.8.1, “QUICC Engine System Interrupt Configuration Register \(CICR\)”](#)).

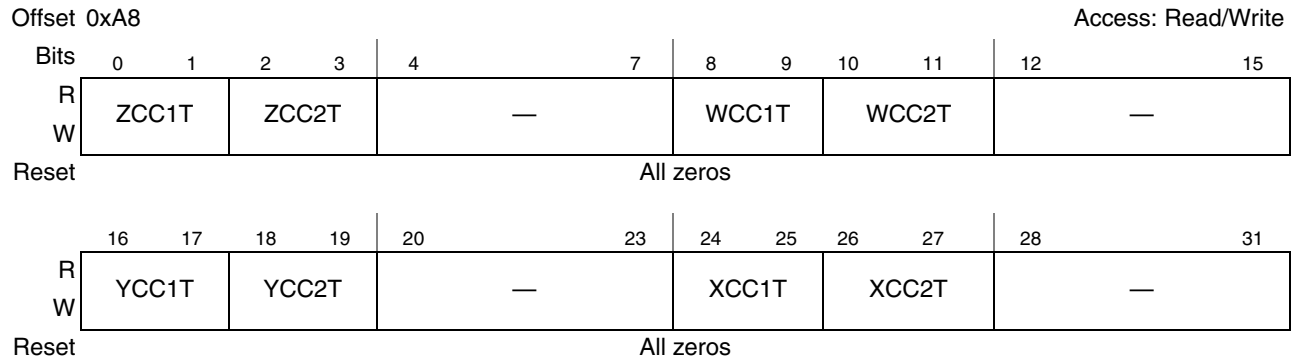


Figure 24-16. QUICC Engine System Internal Interrupt Control Register (CICNR)

[Table 24-13](#) defines the bit fields of CICNR.

Table 24-13. CICNR Bit Settings

Bits	Name	Description
0–1	ZCC1T	ZCC1 priority position output interrupt Type. Defines which type of the output interrupt signal (QUICC Engine High or QUICC Engine Low) asserts its request to the system interrupt controller in the ZCC1 priority position. These bits can not be changed dynamically. (If software really wants to change it, it has to make sure the corresponding interrupt source is masked or it will not happen during the change). The definition of ZCC1T is as follows: 00 QUICC Engine Low request is asserted to the system interrupt controller for ZCC1. 01 Reserved. 10 QUICC Engine High request is asserted to the system interrupt controller for ZCC1. 11 Reserved.
2–3	ZCC2T	Same as ZCC1T, but for ZCC2T.
4–7	—	Write ignored, read = 0
8–9	WCC1T	WCC1 priority position output interrupt Type. Defines which type of the output interrupt signal (QUICC Engine High or QUICC Engine Low) asserts its request to the system interrupt controller in the WCC1 priority position. These bits can not be changed dynamically. (If software really wants to change it, it has to make sure the corresponding interrupt source is masked or it will not happen during the change). The definition of WCC1T is as follows: 00 QUICC Engine Low request is asserted to the system interrupt controller for WCC1. 01 Reserved. 10 QUICC Engine High request is asserted to the system interrupt controller for WCC1. 11 Reserved
10–11	WCC2T	Same as WCC1T, but for WCC2T.
12–15	—	Write ignored, read = 0

Table 24-13. CICNR Bit Settings (continued)

Bits	Name	Description
16–17	YCC1T	YCC1 priority position output interrupt Type. Defines which type of the output interrupt signal (QUICC Engine High or QUICC Engine Low) asserts its request to the system interrupt controller in the YCC1 priority position. These bits can not be changed dynamically. (If software really wants to change it, it has to make sure the corresponding interrupt source is masked or it will not happen during the change). The definition of YCC1T is as follows: 00 QUICC Engine Low request is asserted to the system interrupt controller for YCC1. 01 Reserved. 10 QUICC Engine High request is asserted to the system interrupt controller for YCC1. 11 Reserved
18–19	YCC2T	Same as YCC1T, but for YCC2T.
20–23	—	Write ignored, read = 0
24–25	XCC1T	XCC1 priority position output interrupt Type. Defines which type of the output interrupt signal (QUICC Engine High or QUICC Engine Low) asserts its request to the system interrupt controller in the XCC1 priority position. These bits can not be changed dynamically. (If software really wants to change it, it has to make sure the corresponding interrupt source is masked or it will not happen during the change). The definition of XCC1T is as follows: 00 QUICC Engine Low request is asserted to the system interrupt controller for XCC1. 01 Reserved. 10 QUICC Engine High request is asserted to the system interrupt controller for XCC1. 11 Reserved
26–27	XCC2T	Same as XCC1T, but for XCC2T.
28–31	—	Write ignored, read = 0

24.2.1.8.3 QUICC Engine System RISC Interrupts Control Register (CRICR)

CRICR, shown in [Figure 24-17](#), defines the interrupt output type (QUICC Engine High or QUICC Engine Low) for RISC tasks interrupts.

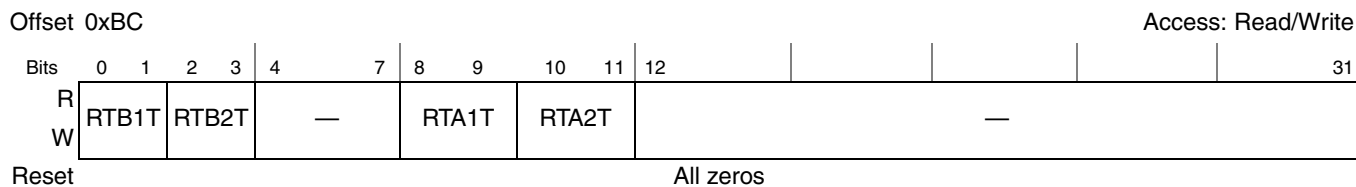


Figure 24-17. QUICC Engine System RISC Interrupts Control Register (CRICR)

[Table 24-14](#) defines the bit fields of CRICR.

Table 24-14. CRICR Bit Settings

Bits	Name	Description
0–1	RTB1T	RTB1 priority position output interrupt Type. Defines which type of the output interrupt signal (QUICC Engine High or QUICC Engine Low) asserts its request to the system interrupt controller in the RTB1 priority position. These bits can not be changed dynamically. (If software really wants to change it, it has to make sure the corresponding interrupt source is masked or it will not happen during the change). The definition of RTB1T is as follows: 00 QUICC Engine Low request is asserted to the system interrupt controller for RTB1. 01 Reserved. 10 QUICC Engine High request is asserted to the system interrupt controller for RTB1. 11 Reserved
2–3	RTB2T	Same as RTB1T, but for RTB2T.
4–7	Reserved	Reserved, should be cleared.
8–9	RTA1T	RTA1 priority position output interrupt Type. Defines which type of the output interrupt signal (QUICC Engine High or QUICC Engine Low) asserts its request to the system interrupt controller in the RTA1 priority position. These bits can not be changed dynamically. (If software really wants to change it, it has to make sure the corresponding interrupt source is masked or it will not happen during the change). The definition of RTA1T is as follows: 00 QUICC Engine Low request is asserted to the system interrupt controller for RTA1. 01 Reserved. 10 QUICC Engine High request is asserted to the system interrupt controller for RTA1. 11 Reserved
10–11	RTA2T	Same as RTA1T, but for RTA2T.
12–31	Reserved	Reserved, should be cleared.

24.2.1.8.4 QUICC Engine System Interrupt Priority Register for WCC Peripherals (CIPWCC)

CIPWCC, shown in [Figure 24-18](#), defines the priority between Timers, SPIs, and external RISC interrupts.

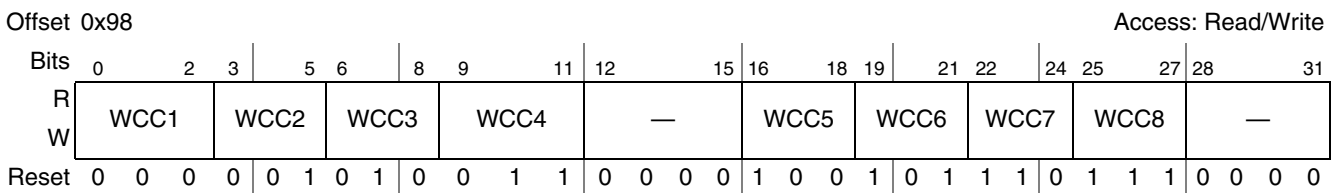


Figure 24-18. QUICC Engine System Interrupt Priority Register for WCC (CIPWCC)

The CIPWCC register bits are described in [Table 24-15](#).

Table 24-15. CIPWCC Field Descriptions

Bits	Name	Description
0–2	WCC1	WCC Priority order. These bits define which QUICC Engine Timer SPI and SDMA error asserts its request in the WCC1 priority position. Note that the same peripheral must not be programmed to more than one priority position (1–8). These bits can be changed dynamically. 000 SPI2 asserts its request in the WCC1 position. 001 SPI1 asserts its request in the WCC1 position. 010 RTT asserts its request in the WCC1 position. 011 Reserved position is not active. 100 Reserved position is not active. 101 Reserved position is not active. 110 Reserved position is not active. 111 Reserved position is not active
3–11, 16–27	WCC2–WCC8	Same as WCC1, but for WCC2–WCC8.
12–15, 28–31	—	Reserved, should be cleared.

NOTE

The lack of SDMA interrupt sources are reported through each individual UCC, MCC, or SPI channel. The SDMA channel bus error entry is the only true SDMA interrupt source that is reported when a bus error occurs during an SDMA access.

24.2.1.8.5 QUICC Engine System Interrupt Priority Register for XCC Peripherals (CIPXCC)

The QUICC Engine system interrupt priority for XCC peripherals register (CIPXCC), shown in [Figure 24-19](#), defines the priorities between the UCCs and MCC.

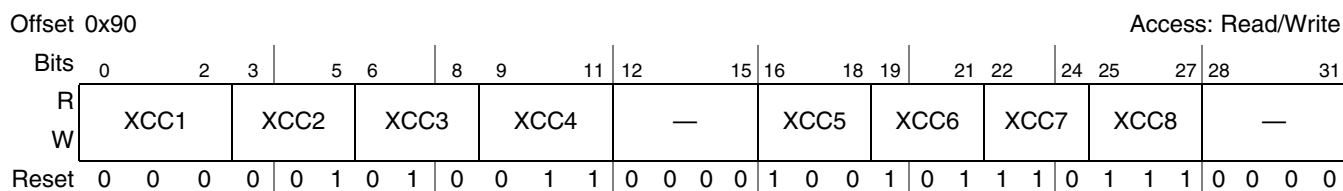


Figure 24-19. QUICC Engine System Interrupt Priority Register for XCC (CIPXCC)

Table 24-16 describes CIPXCC fields.

Table 24-16. CIPXCC Field Descriptions

Bits	Name	Description
0–2	XCC1	Priority order. These bits define which UCC, or the MCC, asserts its request in the XCC1 priority position. Note that each UCC, or the MCC, must not be programmed to more than one priority position (1–8). These bits can be changed dynamically. 000 UCC1 asserts its request in the XCC1 position. 001 UCC2 asserts its request in the XCC1 position. 010 UCC3 asserts its request in the XCC1 position. 011 Reserved position is not active. 100 MCC1 asserts its request in the XCC1 position. 101 Reserved position is not active. 110 Reserved position is not active. 111 Reserved position is not active.
3–11, 16–27	XCC2–XCC8	Same as XCC1, but for XCC2–XCC8
12–15, 28–31	—	Reserved, should be cleared.

24.2.1.8.6 QUICC Engine System Interrupt Priority Register for YCC Peripherals (CIPYCC)

CIPYCC, shown in Figure 24-20, defines the priority of the UCCs.

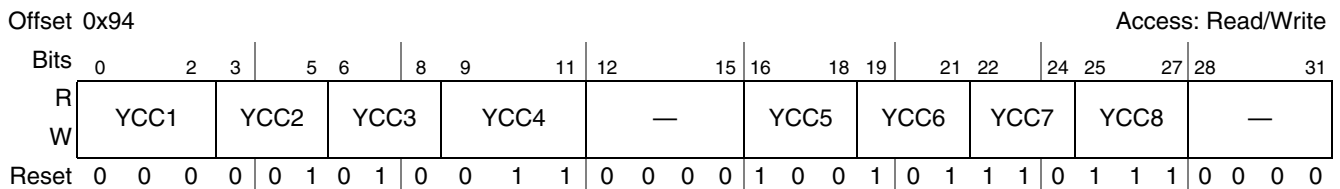


Figure 24-20. QUICC Engine System Interrupt Priority Register for YCC (CIPYCC)

Table 24-17 describes the CIPYCC fields.

Table 24-17. CIPYCC Field Descriptions

Bits	Name	Description
0–2	YCC1	Priority order. These bits define which UCC asserts its request in the YCC1 priority position. The same UCC must not be programmed to multiple priority positions. This field can be changed dynamically. 000 UCC5 asserts its request in the YCC1 position. 001 Reserved position is not active. 010 UCC7 asserts its request in the YCC1 position. 011 Reserved position is not active. 100 101 Reserved position is not active. 110 Reserved position is not active. 111 Reserved position is not active.

Table 24-17. CIPYCC Field Descriptions (continued)

Bits	Name	Description
3–11, 16–27	YCC2–YCC8	Same as YCC1, but for YCC2–YCC8
12–15, 28–31	—	Reserved, should be cleared.

NOTE: On Backward-Compatibility

The CIPXCC register corresponds to the SCPRR_H register in the 85xx PQIII devices. The CIPYCC register corresponds to the SCPRR_L register in the 85xx PQIII devices.

24.2.1.8.7 QUICC Engine System Interrupt Priority Register for ZCC Peripherals (CIPZCC)

CIPZCC, shown in [Figure 24-21](#), defines the priority among a few peripherals.

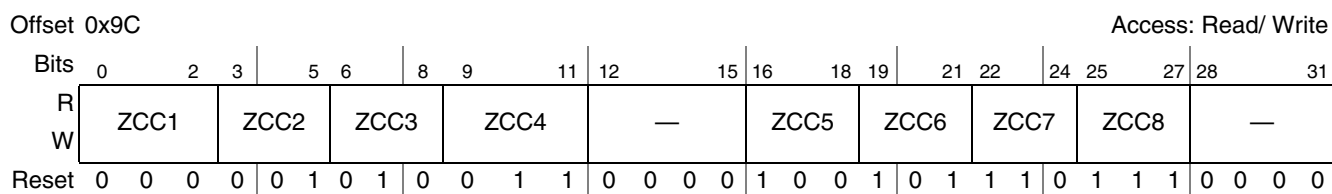


Figure 24-21. QUICC Engine System Interrupt Priority Register for ZCC (CIPZCC)

The CIPZCC register bits are described in [Table 24-18](#).

Table 24-18. CIPZCC Field Descriptions

Bits	Name	Description
0–2	ZCC1	Priority order. These bits define which QUICC Engine Timer and SDMA Sys Error asserts its request in the ZCC1 priority position. The same peripheral must not be programmed to more than one priority position (1–8). These bits can be changed dynamically. 000 Reserved 001 SDMA Sys asserts its request in the ZCC1 position. 010 USB Sys asserts its request in the ZCC1 position. 011 Timer1 asserts its request in the ZCC1 position. 100 Timer2 asserts its request in the ZCC1 position. 101 Timer3 asserts its request in the ZCC1 position. 110 Timer4 asserts its request in the ZCC1 position. 111 Reserved position is not active.
3–11, 16–27	ZCC2–ZCC8	Same as ZCC1, but for ZCC2–ZCC8.
12–15, 28–31	—	Reserved, should be cleared.

24.2.1.8.8 QUICC Engine System Interrupt Priority Register for RISC Tasks A (CIPRTA)

CIPRTA, shown in [Figure 24-22](#) defines the relative priority between the interrupt sources VT, PTP1, PTP2, and RTC.

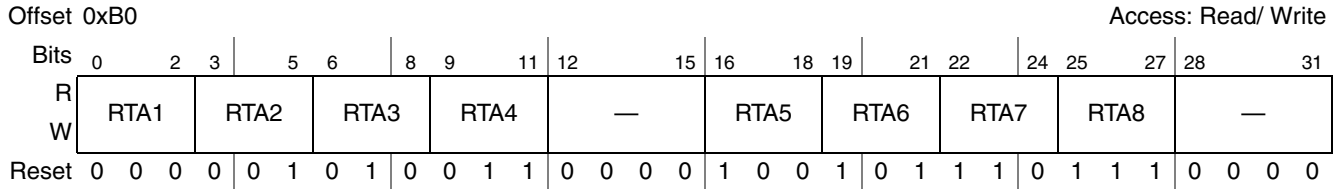


Figure 24-22. QUICC Engine System Interrupt Priority Register for RISC Tasks A (CIPRTA)

The CIPRTA register bits are described in [Table 24-19](#).

Table 24-19. CIPRTA Field Descriptions

Bits	Name	Description
0–2	RTA1	Priority order. These bits define which RISC task A asserts its request in the RTA1 priority position. The same task must not be programmed to more than one priority position (1–8). These bits can be changed dynamically. 000 PTP1 ¹ 001 Reserved 010 PTP2 ¹ 011 Virtual task 100 RTC ¹ 101 Reserved 110 Reserved 111 Reserved
3–11, 16–27	RTA2–RTA8	Same as RTA1, but for RTA2–RTA8.
12–15, 28–31	—	Reserved, should be cleared.

¹ For IEEE 1588 support.

24.2.1.8.9 QUICC Engine System Interrupt Priority Register for RISC Tasks B (CIPRTB)

CIPRTB, shown in [Figure 24-23](#) defines the relative priority among the interrupt sources EXT1, EXT2, EXT3 and EXT4.

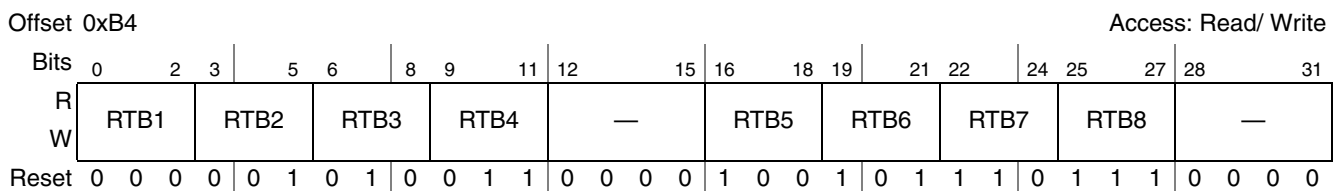


Figure 24-23. QUICC Engine System Interrupt Priority Register for RISC Tasks B (CIPRTB)

The CIPRTB register bits are described in [Table 24-20](#).

Table 24-20. CIPRTB Field Descriptions

Bits	Name	Description
0–2	RTB1	Priority order. These bits define which RISC task B asserts its request in the RTB1 priority position. The same task must not be programmed to more than one priority position (1–8). These bits can be changed dynamically. 000 EXT1 asserts its request in the RTB1 position. 001 EXT2 asserts its request in the RTB1 position. 010 EXT3 asserts its request in the RTB1 position. 011 EXT4 asserts its request in the RTB1 position. 100 Reserved position is not active. 101 Reserved position is not active. 110 Reserved position is not active. 111 Reserved position is not active.
3–11, 16–27	RTB2–RTB8	Same as RTB1, but for RTB2–RTB8.
12–15, 28–31	—	Reserved, should be cleared.

24.2.1.8.10 QUICC Engine System Interrupt Pending Register (CIPNR)

Each bit in the CIPNR, shown in [Figure 24-24](#), corresponds to an interrupt source. When an interrupt is received, the interrupt controller sets the corresponding CIPNR bit.

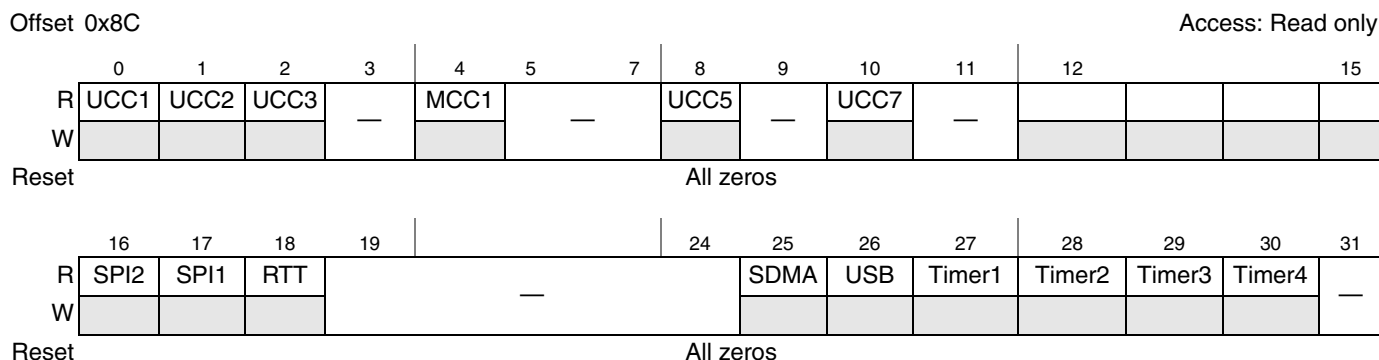


Figure 24-24. QUICC Engine System Interrupt Pending Register (CIPNR)

CIPNR is a read-only register, so when a pending interrupt is handled, the user must clear the corresponding CIPNR bit by clearing the corresponding event register bit. The CIPNR bit position is fixed, and is not affected by the interrupt priority scheme.

24.2.1.8.11 QUICC Engine System Interrupt Mask Register (CIMR)

Each bit in the CIMR, shown in [Figure 24-25](#), corresponds to an interrupt source. The user can mask an interrupt by clearing the relevant bit. Also, an interrupt can be enabled by setting the corresponding CIMR bit. When a masked interrupt occurs, the corresponding CIPNR bit is set, regardless of the CIMR bit although no interrupt request is passed to the core processor.

If an interrupt source requests an interrupt service when the user clears its CIMR bit, the request stops. If the user sets the CIMR bit later, a previously pending interrupt request is processed by the core processor, according to its assigned priority. The CIMR can be read by the user at any time.

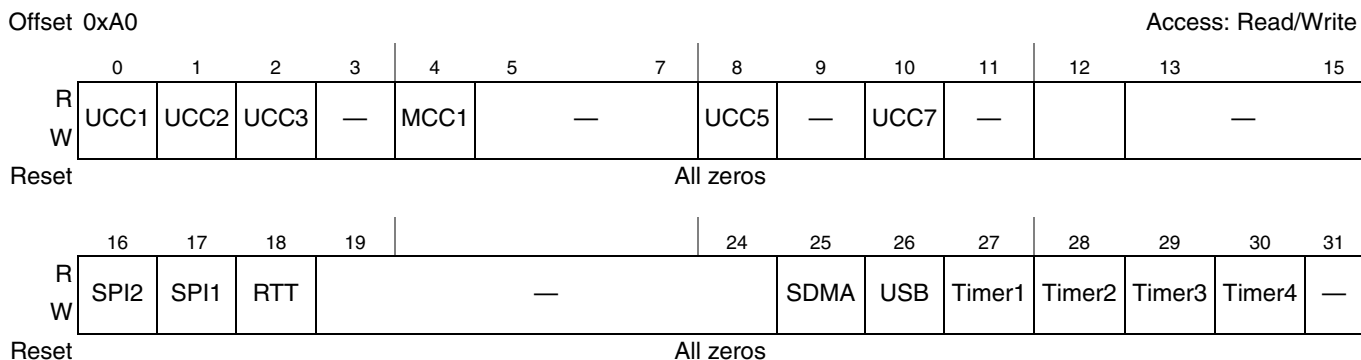


Figure 24-25. CIMR Field

NOTE

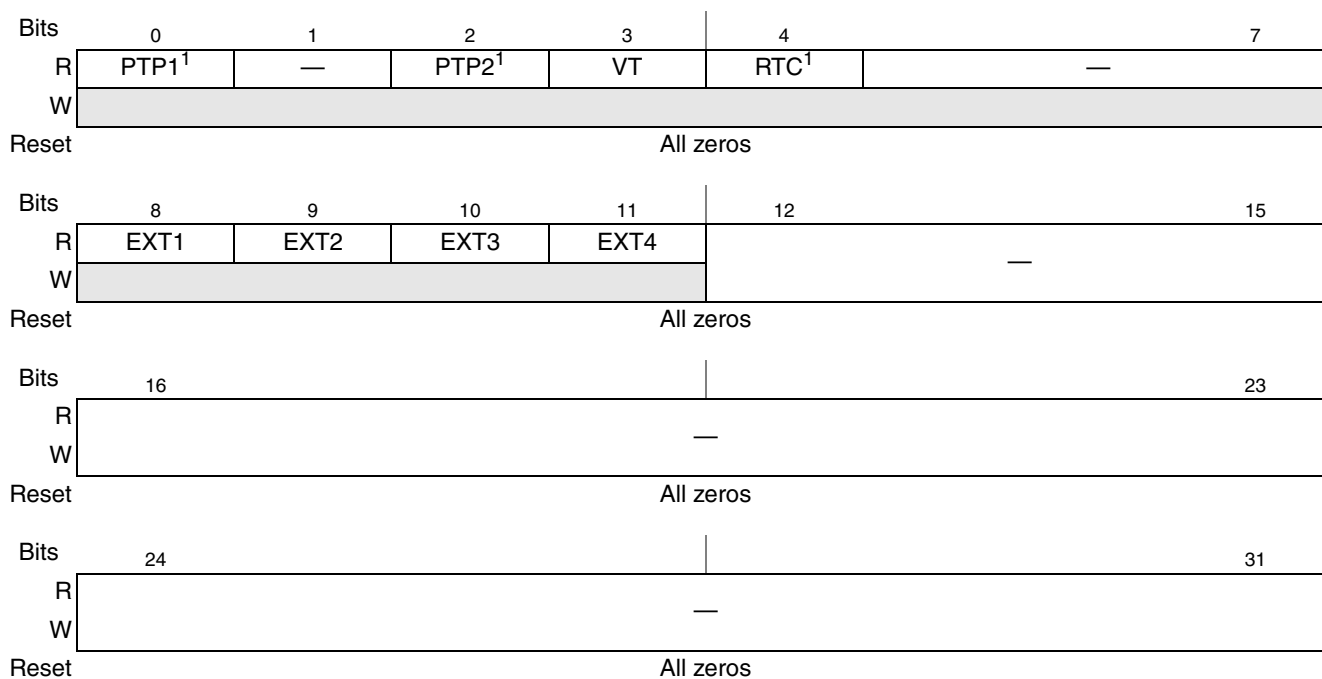
The CIMR bit positions are not affected by the relative interrupt priority. The user can clear pending register bits set by multiple interrupt events only by clearing *all* unmasked events in the corresponding event register. If a CIMR bit is masked at the same time that the corresponding CIPNR bit causes an interrupt request to the core processor, the error vector is issued (if no other interrupts are 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.

24.2.1.8.12 QUICC Engine RISC Interrupt Pending Register (CRIPNR)

Each bit in the QUICC Engine RISC Interrupt Pending Register, represents a pending interrupt in the RISC. For each bit there is a corresponding event register described in the Configuration chapter of the QEIWRM. [Figure 24-26](#) shows the CRIPNR fields.

Offset 0x88

Access: Read Only



¹ These fields are for IEEE 1588 support.

Figure 24-26. CRIPNR Fields

Following is a short description of the RISC interrupt sources:

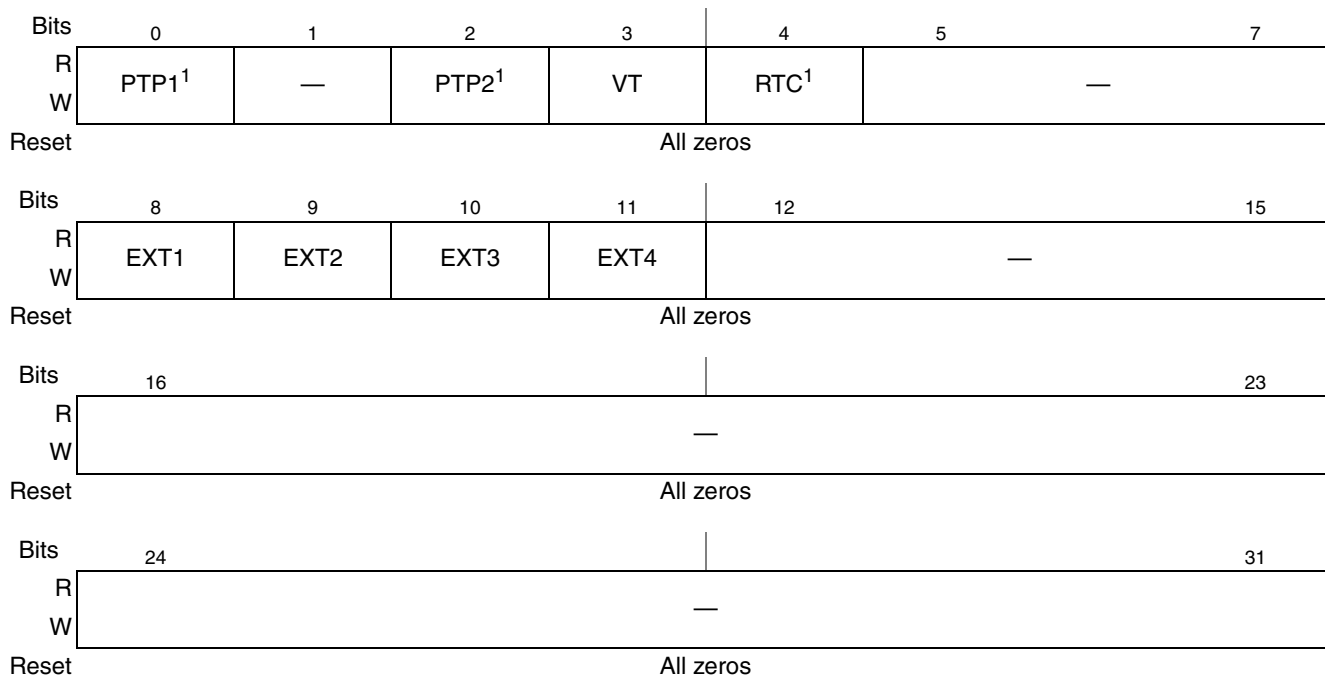
- **IEEE 1588 Interrupt (PTPx, RTC)** are interrupts to the core processor that are asserted by the QUICC Engine module while it is processing an interrupt request from PTPn_TMR_PEVENT or TMR_TEVENT. This mechanism allows for the QUICC Engine RISC to interrupt the host core processor due to certain events which occur while the QUICC Engine UCC's Time Stamp Unit processes a PTP packet or upon detection of an IEEE 1588 Timer event.
- **External Request Interrupt (EXTx)** is an interrupt to the core processor that is asserted by the QUICC Engine module while it is processing an external request. This mechanism allows for the QUICC Engine RISC to interrupt the host core processor due to certain events which occur while the QUICC Engine RISC processes an external request. Up to four external requests are supported by the QUICC Engine module.
- **Virtual Task (VT) Interrupt** is asserted by the RISC when a certain event occur while processing a virtual task. A virtual task is a task that is not directly associated with a peripheral, e.g. a UCC. Up to 28 virtual tasks can run concurrently in the QUICC Engine module. The status bit for the virtual task that is issuing the interrupt is found in the Configuration chapter of the QEIWRM. The exact definition of the assertion of an interrupt in a virtual task is protocol dependent and is not described in this chapter. The virtual tasks on the QUICC Engine module are used to run multiple threads on the UCC Ethernet controller and the ATM controller.

24.2.1.8.13 QUICC Engine RISC Interrupt Mask Register (CRIMR)

CRIMR, shown in [Figure 24-27](#), is used to mask interrupts that are pending in CRIPNR. Setting a bit in CRIMR masks the corresponding pending interrupt in CRIPNR.

Offset 0xA4

Access: Read/Write



¹ These fields are for IEEE 1588 support.

Figure 24-27. CRIMR Fields

24.2.1.8.14 QUICC Engine System Interrupt Vector Register (CIVEC)

CIVEC, shown in [Figure 24-28](#), contains a 6-bit vector code (in bits 0–5) representing the unmasked interrupt source of the highest priority level. The ‘Interrupt Vector Code Image’ field (in bits 26–31) is just a duplication of the value in bits 0–5. The vector in this register corresponds to the group of interrupts which are assigned to the QUICC Engine Low interrupt output. See [Section 24.2.1.7.6, “Interrupt Vector Generation and Calculation,”](#) for the list of codes.

Offset 0x84

Access: Read Only

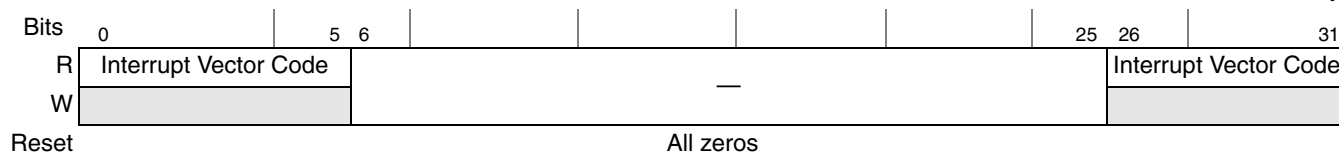


Figure 24-28. QUICC Engine System Interrupt Vector Register (CIVEC)

The CIVEC can be read as either a byte, half word, or a word.

- When read as a byte, a branch table can be used in which each entry contains one instruction (branch).

- When read as a half word, each entry can contain a full routine of up to 256 instructions. The interrupt code is defined such that its two lsb's are zeros, allowing indexing into the table, as shown in [Figure 24-29](#).

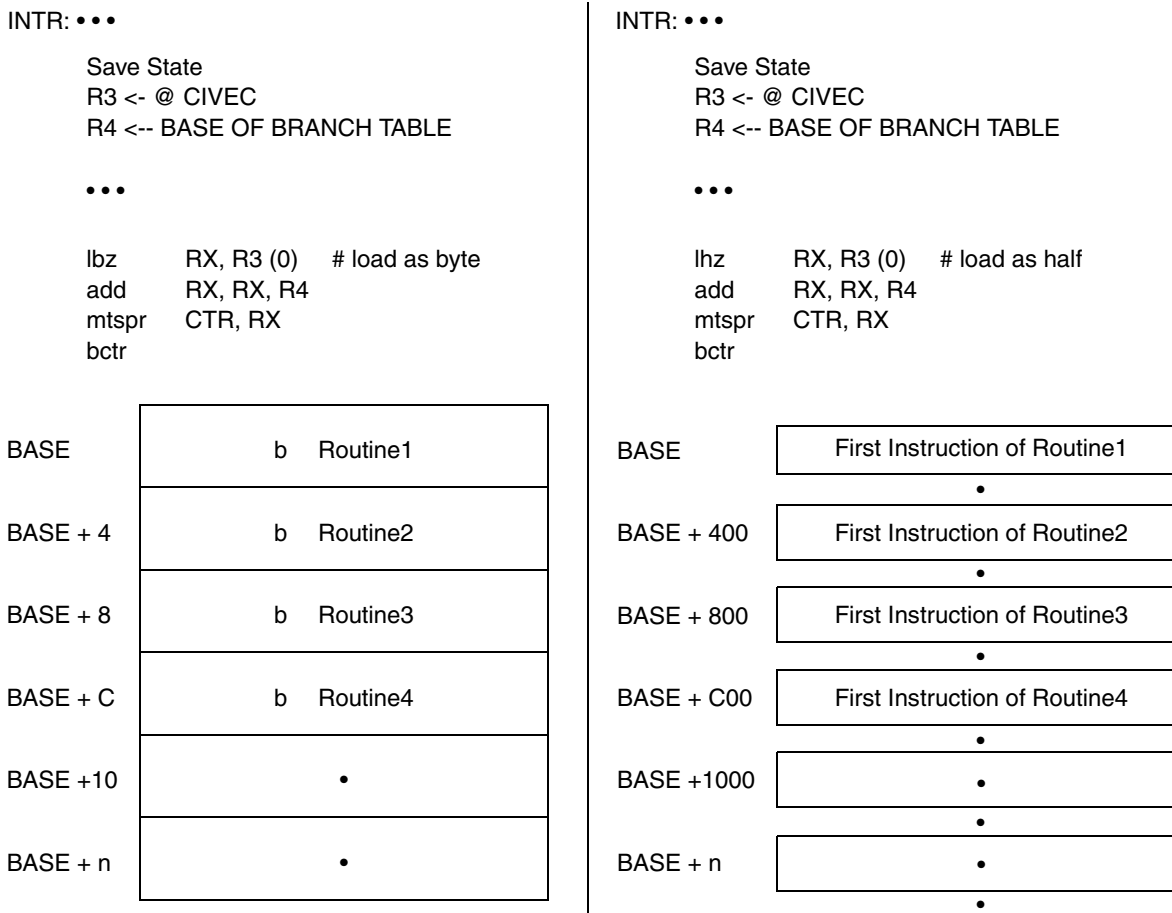


Figure 24-29. Interrupt Table Handling Example

CIVEC can be read when an interrupt request occurs. If there are multiple interrupt sources, CIVEC latches the highest priority interrupt. Note that the value of CIVEC cannot change while it is being read.

24.2.1.8.15 QUICC Engine High System Interrupt Vector Register (CHIVEC)

CHIVEC, shown in [Figure 24-30](#), contains a 6-bit vector code (in bits 0–5) representing the unmasked interrupt source of the highest priority level. The ‘Interrupt Vector Code Image’ field (in bits 26–31) is just a duplication of the value in bits 0–5. The vector in this register corresponds to the group of interrupts which are assigned to the QUICC Engine High interrupt output. See [Section 24.2.1.7.6, “Interrupt Vector Generation and Calculation,”](#) for the list of codes.

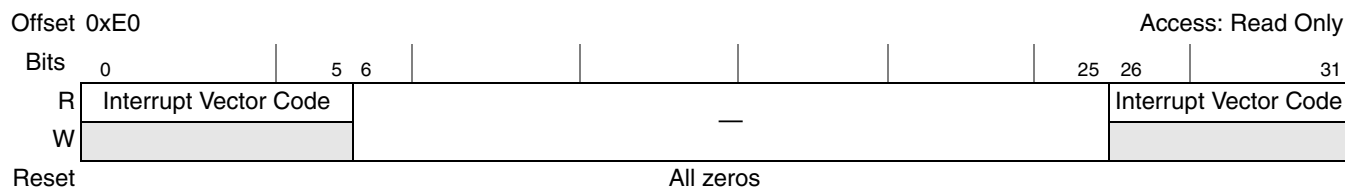


Figure 24-30. QUICC Engine High System Interrupt Vector Register (CHIVEC)

24.2.2 Configuration – Parameter RAM

For MPC8309, the user must override the default values with other values within the multi-user RAM by issuing the ASSIGN PAGE Command to the QUICC Engine block. As the default values are not in the first 16-KB memory space of the multi-user RAM, the user has to modify the page addresses using the assign page host command as part of the initialization process.

Table 24-21 depicts the default values of the parameter RAM Pages base addresses assigned by the QUICC Engine block to the different peripherals.

Table 24-21. Default Parameter RAM Base Addresses

Address	Peripheral	Size (Bytes)
0x8400	UCC1 (Rx and Tx)	256
0x8500	UCC2 (Rx and Tx)	256
0x8600	UCC3 (Rx and Tx)	256
0x8000	UCC5 (Rx and Tx)	256
0x8200	UCC7 (Rx and Tx)	256

As the default values are not in the first 16-KB memory space of the multi-user RAM, the user has to modify the page addresses using the assign page host command as part of the initialization process.

Table 24-22 depicts the suggested values of the parameter RAM Pages base addresses for the different peripherals, but any other valid address can be selected.

Table 24-22. QUICC Engine Parameter RAM Base Addresses (Suggested Value for User Configuration)

Address	Peripheral	Size (Bytes)
0x400	UCC1 (Rx & Tx)	256
0x500	UCC2 (Rx & Tx)	256
0x600	UCC3 (Rx & Tx)	256
0x000	UCC5 (Rx & Tx)	256
0x200	UCC7 (Rx & Tx)	256

24.2.2.1 QUICC Engine Block Control—CERCR[CIR]

In the “QUICC Engine RAM Control Register (CERCR)” subsection, CERCR[CIR] bit is not implemented for the MPC8309.

24.2.2.2 I-RAM Address Register (IADD)

In the “I-RAM Address Register (IADD)” subsection, there is an MPC8309 specific bit field “IADDR.” IADDR is 16 bits; so bits 14 and 15 are reserved.

24.2.2.3 I-RAM Data Register

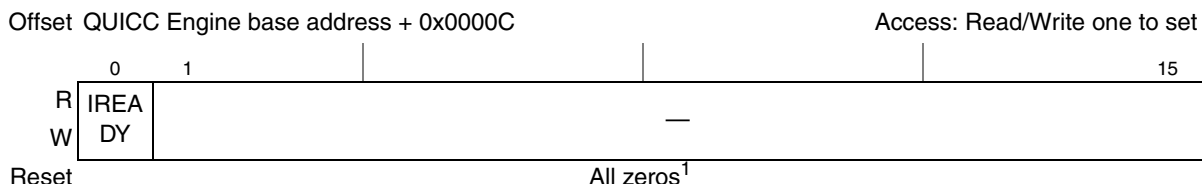
In IRAM Data Register section, this note applies to MPC8309:

NOTE

On the MPC8309, this register is also mapped to a memory-mapped address block at QUICC Engine base address + 0x7000–0x7FFF. If IADDR[IAE]=1, writing to any address in this block results in programming the I-RAM. This enables burst writes to fast programming of the I-RAM therefore speeding up significantly the download of IRAM images.

24.2.2.4 IRAM Ready (IReady) for MPC8309 only

This register must be programmed after downloading microcode in the IRAM of the MPC8309. The QUICC Engine is able to execute the microcode after it is enabled in this register.



¹ This register is not affected by Soft Reset (so microcode does not need to be reloaded after soft reset).

Figure 24-31. I-RAM Ready Register (IReady)

IReady field descriptions are shown in [Table 24-23](#).

Table 24-23. IReady Field Descriptions

Bits	Name	Description
0	IREADY	I-RAM Ready 0 Execution of microcode from I-RAM is disabled 1 Execution of microcode from I-RAM is enabled
1–31	—	Reserved, must be cleared

24.2.2.5 QUICC Engine Microcode Revision Number

In the “QUICC Engine Microcode Revision Number” subsection, the description shows that on the MPC8309, the value for CEURNR[REV_NUM] depends on the loaded microcode package.

24.2.2.6 Serial Number (SNUM)

The SNUM Table in this “Serial Number (SNUM)” subsection includes SNUMs for 12 Internal Virtual Threads: IV Thread 36-IV Thread 47 and 14 External Virtual Threads: EV Thread 00-EV Thread 13.

24.2.2.7 QUICC Engine Block Control—External Requests Device Specific Information

In the “External Requests Device Specific Information” subsection, use this MPC8309-specific information:

The MPC8309 support the connection of a few system level communication peripherals to the QUICC Engine block. The user may configure the connection of the interrupt request signal to be connected either to the Integrated Programmable Interrupt Controller (IPIC), or to be connected to the QUICC Engine block. Routing to the QUICC Engine block allows it to handle the interrupts, thus freeing up the CPU.

The following external requests device specific information applies to MPC8309:

- A bank of 14 external clocks (CLK3 to CLK16) are supported in MPC8309.
- All the BRG clocks (BRG1–BRG11) can be used internally. The following BRG clocks are available as chip outputs: BRG1, BRG2, BRG3, BRG4, BRG5, BRG7, BRG8, BRG9, and BRG11)
- Three Fast Ethernet Controllers on UCC1, UCC2, and UCC3 are available. They can support RMII or MII with 10Mbps or 100Mbps speed.
- One TRB Input (TRB1) is supported
- One TRB Output (TRB1) is supported
- Two CE_PIO[0:1] is supported
- Two TDM ports are supported - TDMA1 (also called TDM1) and TDMA2 (also called TDM2)
- Two Strobes (SI1_STRB[1:2]) are supported
- Two HDLC (HDLC1 and HDLC2) ports are supported. HDLC1 connected to UCC7 and HDLC2 is connected to UCC5.
- For TDM and HDLC the following configurations are supported: 1xTDM1 + 1xHDLC2 (or) 1xTDM2 + 1xHDLC1 (or) 2xTDM (or) 2xHDLC

NOTE

While enabling the HDLC internal loopback, CTS_B to be connected to 0.

- UART is supported on all the 5 UCCs

- For the UART available on UCC1, UCC2 and UCC3
 - CTS_B is available on RX_DV
 - CD_B is available on RX_ER
 - SIN is available on RXD0
 - RTS_B is available on TX_EN
 - SOUT is available on TXD0
- For the UART available on UCC5 and UCC7
 - CTS_B is available on HDLCx_CTS_B
 - CD_B is available on HDLCx_CD_B
 - SIN is available on HDLCx_RXD
 - RTS_B is available on HDLCx_RTS_B
 - SOUT is available on HDLCx_TXD

24.2.2.8 Virtual Threads in MPC8309

In the MPC8309, there are two types of Virtual Threads:

- 14 External Virtual Threads (EV Thread 00-EV Thread 13)
- 12 Internal Virtual Threads (IV Thread 36-IV Thread 47)

The External Virtual Threads are optionally triggered by a Hardware accelerator which is external to the QUICC Engine block. These threads also have extended interrupt capabilities. Each thread has a 32-bit Event/Mask Register used to interrupt the Host CPU. Each thread also has an interrupt pending/mask bit associated with it.

The Internal Virtual Threads, are not triggered by external Hardware, and do not have any interrupt capabilities. See the SNUM Table for allocation of SNUMs to IV Threads.

24.2.2.8.1 External Virtual Threads Event/Mask Registers (CEVTxxER, CEVTxxMR)

Each External Virtual thread has a 32-bit event register (CEVTxxER) and a 32-bit mask register (CEVTxxMR) associated with it. The CEVTxxER can be read at any time and bits are cleared only by writing ones, writing zeros does not affect bit values.

NOTE

This note explains how to correlate between a Virtual Thread's SNUM in the SNUM Table and its corresponding CEVTxxER/CEVTxxMR register pair. Each index xx used in the register name corresponds to the EV Thread xx in SNUM Table. For example: Event Register CEVT00ER/CEVT00MR correspond to EV Thread 00 whose SNUM is 0x88 in SNUM Table.

Offset in Memory Map

- CEVT00ER: QUICC Engine base address + 0x00200 CEVT13ER: QUICC Engine base address + 0x0268
- CEVT16ER: QUICC Engine base address + 0x0270 CEVT29ER: QUICC Engine base address + 0x02D8
- CEVT00MR: QUICC Engine base address + 0x0204..... CEVT13MR: QUICC Engine base address + 0x026C
- CEVT16MR: QUICC Engine base address + 0x0274..... CEVT29MR: QUICC Engine base address + 0x02DC

Note: Indexes 14,15 are skipped yielding a total of 28 pairs of registers.

CEVTxxER Access: Read/Write one to clear event
CEVTxxMR Access: Read / Write

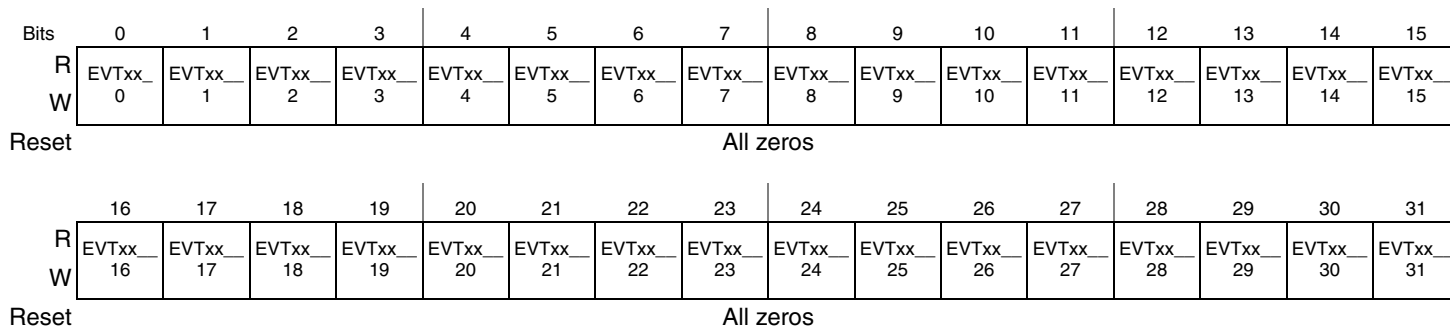


Figure 24-32. External Virtual Thread Event/Mask Register (CEVTxx_ER/CEVTxxMR)

Table 24-24 shows the CEVTxxER field descriptions.

Table 24-24. CEVTxxER Field Descriptions

Bits	Name	Description
0–13	EVTxx- __n	External Virtual Threads 0–13 0 No event detected 1 Event detected
14–31	—	Reserved

Table 24-25 shows the CEVTxxMR field descriptions.

Table 24-25. CEVTxxMR Field Descriptions

Bits	Name	Description
0–13	EVTxx- __n	External Virtual Threads 0–13 0 Interrupt is masked 1 Interrupt is enabled
14–31	—	Reserved

24.2.2.8.2 External Virtual Threads Pending Event/Mask Register (CEVTPER/CEVTPMR)

This register is mapped at the same address as the CEVTER/CEVTMR registers in the MPC8309.

Each CEVTxxER/CEVTxxMR register, has one pending/masking bit associated with it in the CEVTER/CEVTMR Registers. This register is read only.

In addition, the QUICC Engine External Reqeusts [1..4] events have a corresponding pending bit in this register which can be masked.

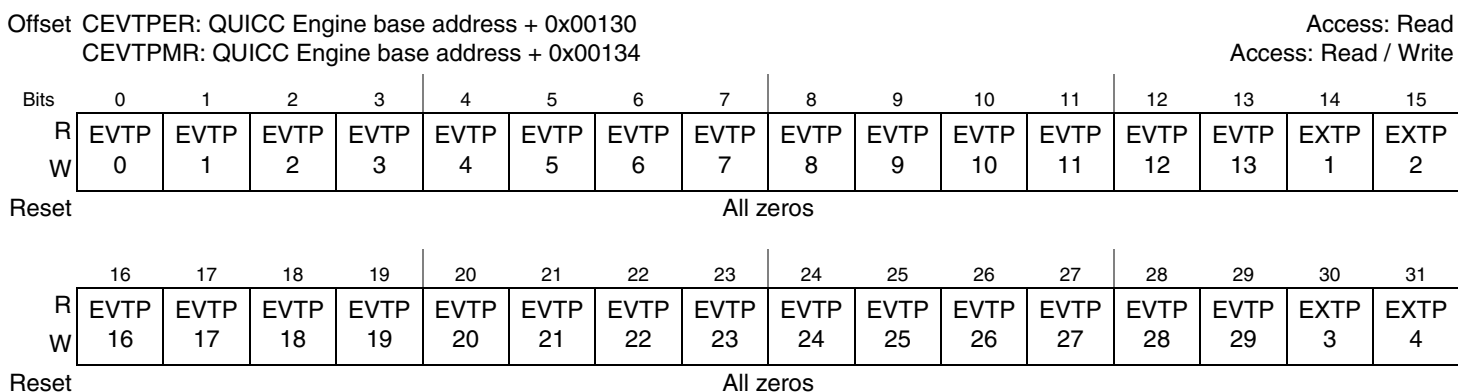


Figure 24-33. Virtual Thread Pending Event/Mask Register (CEVTPER/CEVTPMR)

Table 24-26 shows the CEVTPER field descriptions.

Table 24-27 shows the CEVTPMR field descriptions.

Table 24-26. CEVTPER Field Descriptions

Bits	Name	Description
0–13	EVTP _n	External Virtual Threads 0–13 Pending Interrupt in corresponding CEVTxxER register 0 No event pending 1 Event pending
14–31	—	Reserved

Table 24-27. CEVTPMR Field Descriptions

Bits	Name	Description
0–13	EVTP _n	External Virtual Threads 0–13 Interrupt Pending Mask 0 Interrupt is masked 1 Interrupt is enabled
14–31	—	Reserved

24.2.3 QUICC Engine Multiplexing and Timers

The information in this MPC8309-specific “QUICC Engine Multiplexing and Timers” subsection applies to the “QUICC Engine Multiplexing and Timers” chapter of the QEIWRM.

- Use any information about the following, which do apply to the MPC8309.
 - A bank of 14 external clocks
 - x2 TDM

- TDMA1 and TDMA2
- Disregard any information about the following, which do not apply to the MPC8309.
 - UCC1 GRX CLK
 - UCC2 GRX CLK
 - UCC1 TBI RX CLK
 - UCC2 TBI RX CLK

The QUICC Engine multiplexing and timers logic (CMX) routes clocks and connects the physical interfaces (such as modem lines, TDM lines and proprietary serial lines) to the QUICC Engine peripherals (UCC's, TDM's, etc.).

The CMX has the following key functions:

- The CMX routes clocks to all the QUICC Engine peripherals from a bank of internal clocks (BRGs 1-5,7,8,9,11) and a bank of external clocks (3-16) from the parallel I/O ports.
- Per UCC, the CMX works in either NMSI or TDM mode. In NMSI mode, the CMX allows all peripherals to be connected to their own individual pins. In TDM mode, the CMX connects the

UCCs to the serial interface. This enables the UCCs to use the time-slot assigner (TSA) which allows the UCCs to multiplex data on the TDM channels.

- The CMX selects which of the UCCs is chosen as serial management interface (SMI) master.

Figure 24-34 shows a block diagram of the CMX.

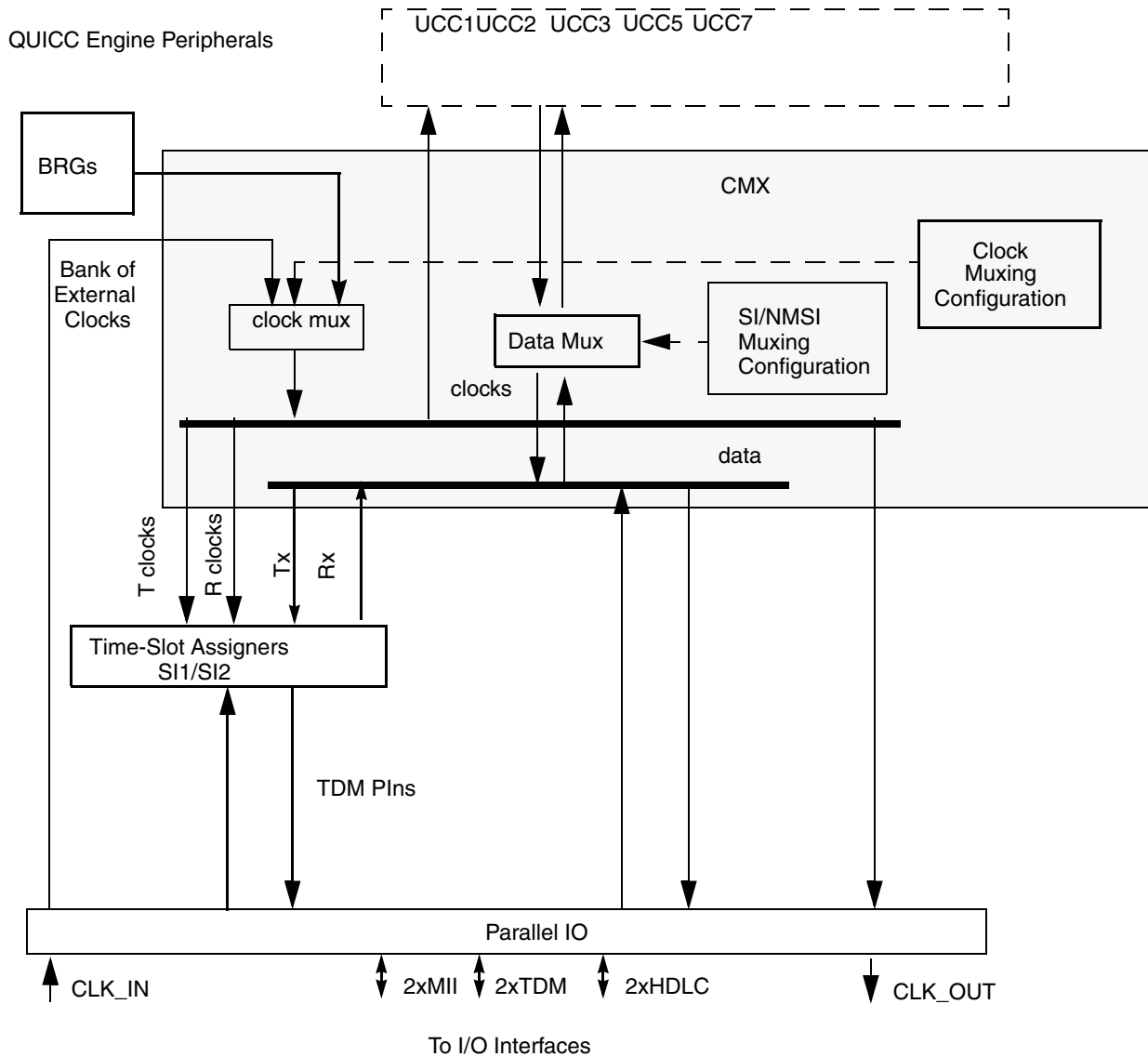


Figure 24-34. QUICC Engine Multiplexing and Timers Logic (CMX) Block Diagram

24.2.3.1 Working with the TDM

When working with TDM, the CMX requires clock and sync to be assigned to the SI. See specific details in the CMXSI1CRL, CMXSI1CRH and CMXSI1SYR registers for SI1 and CMXSI2CRL, CMXSI2CRH and CMXSI2SYR registers for SI2. The clocks and syncs can be assigned from an external bank of 14 clock pins or a bank of 16 BRGs to TDMA1 and TDMA2.

24.2.3.2 QUICC Engine Multiplexing and Timers—CMXUCR n

In the QUICC Engine multiplexing and timers logic (CMX) UCC Clock Route (CMXUCR) registers, the HDLC bus mode fields [HBM n] vary by device. The location for CMXUCR n [HBM n] on the MPC8309 follow this pattern of as shown in the CMXUCR1 example using bit fields 2 and 18 in [Figure 24-35](#).

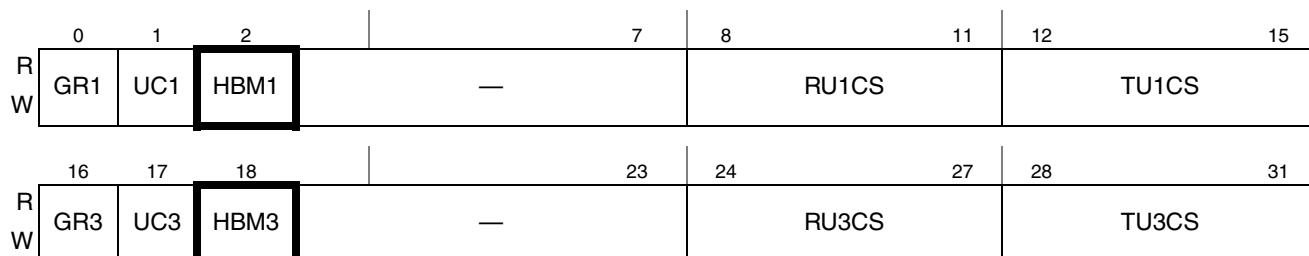


Figure 24-35. CMXUCR1[HBM1] and CMXUCR1[HBM3] location on MPC8309

For the MPC8309, the locations for the HBM n fields are as follows for the four CMXUCR n registers:

- CMXUCR1[HBM1] is CMXUCR1[2] as shown in [Figure 24-35](#).
- CMXUCR1[HBM3] is CMXUCR1[18] as shown in [Figure 24-35](#).
- CMXUCR2[HBM5] is CMXUCR2[2].
- CMXUCR2[HBM7] is CMXUCR2[18].
- CMXUCR3[HBM2] is CMXUCR3[2].
- CMXUCR3[HBM4] is CMXUCR3[18].
- CMXUCR4[HBM6] is CMXUCR4[2].
- CMXUCR4[HBM8] is CMXUCR4[18].

24.2.4 UCC Ethernet (UEC)

The information in this MPC8309-specific “UCC Ethernet (UEC)” subsection applies throughout the QEIWRM “UCC Ethernet (UEC)” chapter.

- MIIGSK1 and MIIGSK2 not supported
- Support for MII and RMII only
- No Burst Tolerance support
- No L2 cache and L2 cache stashing
- SMII and SGMII information does not apply to the MPC8309
- RxDiscOV is a 32-bit counter on the MPC8309

24.2.5 IEEE Standard 1588 Assist

The information in this MPC8309-specific “QUICC Engine IEEE Std. 1588 Assist” subsection applies throughout the QEIWRM “UCC Ethernet (UEC)” chapter.

- Supports IEEE 1588 V2



Appendix A

Complete List of Configuration, Control, and Status Registers

A.1 Local Access Windows

Table A-1. Local Access Register Memory Map

Local Access—Block Base Address 0x0_0000				
Local Memory Offset (Hex)	Register	Access	Reset	Section/Page
0x0_0000	Internal memory map base address register (IMMRBAR)	R/W	0xFF40_0000	6.2.4.1/6-6
0x0_0004	Reserved	—	—	—
0x0_0008	Alternate configuration base address register (ALTCBAR)	R/W	0x0000_0000	6.2.4.2/6-7
0x0_000C–0x0_001C	Reserved	—	—	—
0x0_0020	eLBC local access window 0 base address register (LBLAWBAR0)	R/W	0x0000_0000 ¹	6.2.4.3/6-8
0x0_0024	eLBC local access window 0 attribute register (LBLAWAR0)	R/W	0x0000_0000 ²	6.2.4.4/6-9
0x0_0028	eLBC local access window 1 base address register (LBLAWBAR1)	R/W	0x0000_0000	6.2.4.3/6-8
0x0_002C	eLBC local access window 1 attribute register (LBLAWAR1)	R/W	0x0000_0000	6.2.4.4/6-9
0x0_0030	eLBC local access window 2 base address register (LBLAWBAR2)	R/W	0x0000_0000	6.2.4.3/6-8
0x0_0034	eLBC local access window 2 attribute register (LBLAWAR2)	R/W	0x0000_0000	6.2.4.4/6-9
0x0_0038	eLBC local access window 3 base address register (LBLAWBAR3)	R/W	0x0000_0000	6.2.4.3/6-8
0x0_003C	eLBC local access window 3 attribute register (LBLAWAR3)	R/W	0x0000_0000	6.2.4.4/6-9
0x0_0040–0x0_005C	Reserved	—	0x0000_0000	—
0x0_0060	PCI local access window 0 base address register (PCILAWBAR0)	R/W	0x0000_0000 ³	6.2.4.5/6-10
0x0_0064	PCI local access window 0 attribute register (PCILAWAR0)	R/W	0x0000_0000 ⁴	6.2.4.6/6-11
0x0_0068	PCI local access window 1 base address register (PCILAWBAR1)	R/W	0x0000_0000 ⁵	6.2.4.5/6-10
0x0_006C	PCI local access window 1 attribute register (PCILAWAR1)	R/W	0x0000_0000	6.2.4.6/6-11
0x0_0070–0x0_00	Reserved	—	—	—
0x0_00A0	DDR2 local access window 0 base address register (DDRLAWBAR0)	R/W	0x0000_0000 ⁶	6.2.4.7/6-12

Table A-1. Local Access Register Memory Map (continued)

Local Access—Block Base Address 0x0_0000				
Local Memory Offset (Hex)	Register	Access	Reset	Section/Page
0x0_00A4	DDR2 local access window 0 attribute register (DDRLAWAR0)	R/W	0x0000_0000 ⁷	6.2.4.8/6-13
0x0_00A8	DDR2 local access window 1 base address register (DDRLAWBAR1)	R/W	0x0000_0000	6.2.4.7/6-12
0x0_00AC	DDR2 local access window 1 attribute register (DDRLAWAR1)	R/W	0x0000_0000	6.2.4.8/6-13
0x0_00B0–0x0_00FC	Reserved	—	—	—

- ¹ Depends on reset configuration word high values. See [Section 6.2.4.3.1, “LBLAWBAR0\[BASE_ADDR\] Reset Value,”](#) for details.
- ² Depends on reset configuration word high values. See [Section 6.2.4.4.1, “LBLAWAR0\[EN\] and LBLAWAR0\[SIZE\] Reset Value,”](#) for details.
- ³ Depends on reset configuration word high values. See ” for details.
- ⁴ Depends on reset configuration word high values. See ” for details.
- ⁵ Depends on reset configuration word high values. See [Section 6.2.4.6.1, “PCILAWAR0\[EN\] and PCILAWAR0\[SIZE\] Reset Value,”](#) for details.
- ⁶ Depends on reset configuration word high values. See [Section 6.2.4.7.1, “DDRLAWBAR0\[BASE_ADDR\] Reset Value,”](#) for details.
- ⁷ Depends on reset configuration word high values. See [Section 6.2.4.8.1, “DDRLAWAR0\[EN\] and DDRLAWAR0\[SIZE\] Reset Value,”](#) for details.

A.2 System Configuration Registers

Table A-2. System Configuration Register Memory Map

Local Memory Offset (Hex)	Register	Access	Reset	Section/Page
System Configuration—Block Base Address 0x0_0000				
0x00100	System general purpose register low (SGPRL)	R/W	0x480E_FF20 or 0x0000_0000	6.3.2.1/6-17
0x00104	System general purpose register high (SGPRH)	R/W	0x0000_0000	6.3.2.2/6-18
0x00108	System part and revision ID register (SPRIDR)	R	0x8110_0010	6.3.2.3/6-18
0x0010C	Reserved	—	—	—
0x00110	System priority configuration register (SPCR)	R/W	0x0000_0000	6.3.2.4/6-19
0x00114	System I/O configuration register 1 (SICR_1)	R/W	0x0000_000A (other-boot) 0x0028_000A (esdhc-boot)	6.3.2.5/6-21
0x00118	System I/O configuration register 2 (SICR_2)	R/W	0x9040_5500 (PCIARB=1) 0x9042_5500 (PCIARB=0)	6.3.2.6/6-24

Table A-2. System Configuration Register Memory Map (continued)

Local Memory Offset (Hex)	Register	Access	Reset	Section/Page
0x00128	DDR control driver register (DDRCDR)	R/W	All zeros	6.3.2.9/6-29
0x0012C	DDR debug status register (DDRDSR)	R	0x3300_0000	6.3.2.10/6-30
0x00144	eSDHC Control Register (SDHCCR)	R/W	0x0000_0000	6.3.2.11/66-30
0x00148	CAN access control register (CAN_DBG_CTRL)	R/W	0x0000_0000	6.3.2.12/66-32
0x0014C	SPI chip select register (SPI_CS)	R/W	0x0000_0000	6.3.2.13/66-33
0x00150	General purpose register 1 (GPR_1)	R/W	0x0080_8001	6.3.2.14/66-34
0x00154–0x00167	Reserved	—	—	—
0x168	SIDCR2	R/W	0x0000_0000 (M66EN=0) 0xFC00_0000 (M66EN=1)	6.3.2.16/66-36
0x00169–0x001BC	Reserved	—	—	—
0x001C0	CAN interrupt status register (CAN_INT_STAT)	R	0x0000_0000	6.3.2.16/66-36
0x001C4	DUART interrupt status register (DUART_INT_STAT)	R	0x0000_0000	6.3.2.17/66-38
0x001C8	GPIO interrupt status register (GPIO_INT_STAT)	R	0x0000_0000	6.3.2.18/66-39
0x001CC–0x001FC	Reserved	—	—	—

A.3 Watchdog Timer (WDT)

Table A-3. WDT Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
Watchdog Timer (WDT)—Block Base Address 0x0_0200				
0x00–0x03	Reserved	—	—	—
0x004	System watchdog control register (SWCRR)	R/W	0xFFFF_0003 or 0xFFFF_0007 ¹	6.4.4.1/6-42
0x008	System watchdog count register (SWCNR)	R	0x0000_FFFF	6.4.4.2/6-43
0x00C–0x00D	Reserved	—	—	—
0x00E	System watchdog service register (SWSRR)	R/W	0x0000	6.4.4.3/6-43

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

A.4 Real Time Clock (RTC)

Table A-4. RTC Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
Real Time Clock (RTC)—Block Base Address 0x0_0300				
0x00	Real time counter control register (RTCNR)	R/W	0x0000_0000	6.5.5.1/6-49
0x04	Real time counter load register (RTLDR)	R/W	0x0000_0000	6.5.5.2/6-49
0x08	Real time counter prescale register (RTPSR)	R/W	0x0000_0000	6.5.5.3/6-50
0x0C	Real time counter register (RTCTR)	R	0x0000_0000	6.5.5.4/6-50
0x10	Real time counter event register (RTEVR)	w1c	0x0000_0000	6.5.5.5/6-51
0x14	Real time counter alarm register (RTALR)	R/W	0xFFFF_FFFF	6.5.5.6/6-51
0x18–0x1F	Reserved	—	—	

A.5 Periodic Interval Timer (PIT)

Table A-5. PIT Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
Periodic Interval Timer (PIT)—Block Base Address 0x0_0400				
0x000	Periodic interval timer control register (PTCNR)	R/W	0x0000_0000	6.6.5.1/6-56
0x004	Periodic interval timer load register (PTLDR)	R/W	0x0000_0000	6.6.5.2/6-56
0x008	Periodic interval timer prescale register (PTPSR)	R/W	0x0000_0000	6.6.5.3/6-57
0x00C	Periodic interval timer counter register (PTCTR)	R	0x0000_0000	6.6.5.4/6-57
0x10	Periodic interval timer event register (PTEVR)	w1c	0x0000_0000	6.6.5.5/6-58
0x014–0x01F	Reserved	—	—	

A.6 General Purpose (Global) Timers (GTMs)

Table A-6. GTM Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
General Purpose (Global) Timer Module 1—Block Base Address 0x0_0500 General Purpose (Global) Timer Module 1—Block Base Address 0x0_0600				
0x000	Timer 1 and 2 global timers configuration register (GTCFR1)	R/W	0x00	6.7.5.1/6-65
0x001–0x003	Reserved	—	—	—
0x004	Timer 3 and 4 global timers configuration register (GTCFR2)	R/W	0x00	6.7.5.1/6-65
0x005–0x00F	Reserved	—	—	—
0x010	Timer 1 global timers mode register (GTMDR1)	R/W	0x0000	6.7.5.2/6-69
0x012	Timer 2 global timers mode register (GTMDR2)			

Table A-6. GTM Register Address Map (continued)

Offset	Register	Access	Reset Value	Section/ Page
0x014	Timer 1 global timers reference register (GTRFR1)	R/W	0xFFFF	6.7.5.3/6-70
0x016	Timer 2 global timers reference register (GTRFR2)			
0x018	Timer 1 global timers capture register (GTCPR1)	R/W	0x0000	6.7.5.4/6-70
0x01A	Timer 2 global timers capture register (GTCPR2)			
0x01C	Timer 1 global timers counter register (GTCNR1)	R/W	0x0000	6.7.5.5/6-71
0x01E	Timer 2 global timers counter register (GTCNR2)			
0x020	Timer 3 global timers mode register (GTMDR3)	R/W	0x0000	6.7.5.2/6-69
0x022	Timer 4 global timers mode register (GTMDR4)			
0x024	Timer 3 global timers reference register (GTRFR3)	R/W	0xFFFF	6.7.5.3/6-70
0x026	Timer 4 global timers reference register (GTRFR4)			
0x028	Timer 3 global timers capture register (GTCPR3)	R	0x0000	6.7.5.4/6-70
0x02A	Timer 4 global timers capture register (GTCPR4)			
0x02C	Timer 3 global timers counter register (GTCNR3)	R/W	0x0000	6.7.5.5/6-71
0x02E	Timer 4 global timers counter register (GTCNR4)			
0x030	Timer 1 global timers event register (GTEVR1)	w1c	0x0000	6.7.5.6/6-71
0x032	Timer 2 global timers event register (GTEVR2)			
0x034	Timer 3 global timers event register (GTEVR3)			
0x036	Timer 4 global timers event register (GTEVR4)			
0x038	Timer 1 global timers prescale register (GTPSR1)	R/W	0x0003	6.7.5.7/6-72
0x03A	Timer 2 global timers prescale register (GTPSR2)			
0x03C	Timer 3 global timers prescale register (GTPSR3)			
0x03E	Timer 4 global timers prescale register (GTPSR4)			
General Purpose (Global) Timer Module 2: All registers defined for GTM1 are also defined for GTM2; the base address of GTM2 registers is 0x0_0600. The offset for the registers of Global Timer Module 2 timers (GTM 5-8) remains the same.				

A.7 Integrated Programmable Interrupt Controller (IPIC)

Table A-7. IPIC Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
Integrated Programmable Interrupt Controller—Block Base Address 0x0_0700				
0x00	System global interrupt configuration register (SICFR)	R/W	0x0000_0000	9.5.1/9-7
0x04	System regular interrupt vector register (SIVCR)	R	0x0000_0000	9.5.2/9-8
0x08	System internal interrupt pending register (SIPNR_H)	R	0x0000_0000	9.5.3/9-11

Table A-7. IPIC Register Address Map (continued)

Offset	Register	Access	Reset Value	Section/ Page
0x0C	System internal interrupt pending register (SIPNR_L)	R	0x0000_0000	9.5.3/9-11
0x10	System internal interrupt group A priority register (SIPRR_A)	R/W	0x0530_9770	9.5.4/9-14
0x14	System internal interrupt group B priority register (SIPRR_B)	R/W	0x0530_9770	9.5.5/9-15
0x18	System internal interrupt group C priority register (SIPRR_C)	R/W	0x0530_9770	9.5.6/9-15
0x1C	System internal interrupt group D priority register (SIPRR_D)	R/W	0x0530_9770	9.5.7/9-16
0x20	System internal interrupt mask register (SIMSR_H)	R/W	0x0000_0000	9.5.8/9-17
0x24	System internal interrupt mask register (SIMSR_L)	R/W	0x0000_0000	9.5.8/9-17
0x28	System internal interrupt control register (SICNR)	R/W	0x0000_0000	9.5.9/9-18
0x2C	System external interrupt pending register (SEP NR)	R/W	Special	9.5.10/9-20
0x30	System mixed interrupt group A priority register (SMPRR_A)	R/W	0x0530_9770	9.5.11/9-21
0x34	System mixed interrupt group B priority register (SMPRR_B)	R/W	0x0530_9770	9.5.12/9-22
0x38	System external interrupt mask register (SEMSR)	R/W	0x0000_0000	9.5.13/9-22
0x3C	System external interrupt control register (SECNR)	R/W	0x0000_0000	9.5.14/9-23
0x40	System error status register (SERSR)	R/W	0x0000_0000	9.5.15/9-25
0x44	System error mask register (SERMR)	R/W		9.5.16/9-26
0x48	System error control register (SERCR)	R/W	0x0000_0000	9.5.17/9-27
0x4C	System external interrupt polarity control register (SEPCR)	R/W	0x0000_0000	Figure 24-3 0./24-35
0x4F	Reserved	—	—	—
0x50	System internal interrupt force register (SIFCR_H)	R/W	0x0000_0000	9.5.19/9-28
0x54	System internal interrupt force register (SIFCR_L)	R/W	0x0000_0000	9.5.19/9-28
0x58	System external interrupt force register (SEFCR)	R/W	0x0000_0000	9.5.20/9-30
0x5C	System error force register (SERFR)	R/W	0x0000_0000	9.5.21/9-30
0x60	System critical interrupt vector register (SCVCR)	R	0x0000_0000	9.5.22/9-31
0x64	System management interrupt vector register (SMVCR)	R	0x0000_0000	9.5.23/9-31
0x68–0xBF	Reserved	—	—	—

A.8 QUICC Engine Ports Interrupts

Table A-8. QUICC Engine Ports Interrupts Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
0x0C	QUICC Engine ports interrupt event register (CEPIER)	w1c	Special	21.3.4/21-5
0x10	QUICC Engine ports interrupt mask register (CEPIMR)	R/W	0x0000_0000	9.5.25/9-33
0x14	QUICC Engine ports interrupt control register (CEPICR)	R/W	0x0000_0000	21.3.6/21-6

A.9 System Arbiter

Table A-9. Arbiter Register Map

System Arbiter—Block Base Address 0x0_0800				
Memory Offset (Hex)	Register	Access	Reset	Section/Page
0x00	Arbiter configuration register (ACR)	R/W	0x0000_0000/ 0x0010_0000 ¹	7.2.1/7-3
0x04	Arbiter timers register (ATR)	R/W	FFFF_FFFF	7.2.2/7-4
0x0C	Arbiter event register (AER)	w1c	0x0000_0000	7.2.3/7-5
0x10	Arbiter interrupt definition register (AIDR)	R/W	0x0000_0000	7.2.4/7-6
0x14	Arbiter mask register (AMR)	R/W	0x0000_0000	7.2.5/7-7
0x18	Arbiter event attributes register (AEATR)	R	0x0000_0000 ²	7.2.6/7-8
0x1C	Arbiter event address register (AEADR)	R	0x0000_0000	7.2.7/7-9
0x20	Arbiter event response register (AERR)	R/W	0x0000_0000	7.2.8/7-10

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 $\overline{\text{PORESET}}$.

A.10 Reset Configuration

Table A-10. Reset Configuration and Status Registers Memory Map

Address	Register	Access	Reset	Section/Page
Reset Configuration—Block Base Address 0x0_0900				
0x0_0900	Reset configuration word low register (RCWLR)	R	0x0000_0000	4.5.1.1/4-27
0x0_0904	Reset configuration word high register (RCWHR)	R	0x0000_0000	4.5.1.2/4-27
0x0_0908– 0x0_090C	Reserved, should be cleared	—	—	—
0x0_0910	Reset status register (RSR)	R/W	0x0000_000	4.5.1.3/4-28

Table A-10. Reset Configuration and Status Registers Memory Map (continued)

Address	Register	Access	Reset	Section/Page
0x0_0914	Reset mode register (RMR)	R/W	0x0000_0000	4.5.1.4/4-29
0x0_0918	Reset protection register (RPR)	R/W	0x0000_0000	4.5.1.5/4-30
0x0_091C	Reset control register (RCR)	R/W	0x0000_0000	4.5.1.6/4-30
0x0_0920	Reset control enable register (RCER)	R/W	0x0000_0000	4.5.1.7/4-31
0x0_0924	Reserved.	—	0xFFFF_FFFF	—
0x0_0928– 0x0_09FC	Reserved, should be cleared	—	0x0000_0000	—

A.11 Clock Configuration

Table A-11. Clock Configuration Registers Memory Map

Address	Register	Access	Reset	Section/Page
Reset Configuration—Block Base Address 0x0_0A00				
0x0_0A00	System PLL mode register (SPMR)	R	0xnnnn_nnnn	4.5.2.1/4-32
0x0_0A04	Output clock control register (OCCR)	R/W	0x0000_C0C0	4.5.2.2/4-33
0x0_0A08	System clock control register (SCCR)	R/W	0x0542_0010	4.5.2.3/4-34
0x0_0A0C– 0x0_0AFC	Reserved, should be cleared	—	—	—

A.12 Power Management Controller (PMC)

Table A-12. Power Management Controller Registers Memory Map

Offset	Register	Access	Reset	Section/Page
PMC—Block Base Address 0x0_0B00				
0x00B00	Power management controller configuration register (PMCCR)	R/W	0x0000_0000	6.8.2.1/6-77
0x00B04–0x00 BFC	Reserved	—	—	—

A.13 General Purpose I/O (GPIO)

Table A-13. GPIO Register Address Map

Offset	Register	Access	Reset Value	Section/Page
General Purpose I/O (GPIO1)—Block Base Address 0x0_0C00 General Purpose I/O (GPIO2)—Block Base Address 0x0_0D00				
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

Table A-13. GPIO Register Address Map (continued)

Offset	Register	Access	Reset Value	Section/Page
0xC08	GPIO1 data register (GP1DAT)	R/W	0x0000_0000	21.3.3/21-4
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)	w1c	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	—	—	—

A.14 DDR Memory Controller

Table A-14. DDR Memory Controller Memory Map

Offset	Register	Access	Reset	Section/Page
0x000	CS0_BNDS—Chip select memory bounds	R/W	0x0000_0000	10.4.1.1/10-11
0x080	CS0_CONFIG—Chip select configuration	R/W	0x0000_0000	10.4.1.2/10-11
0x100	TIMING_CFG_3—DDR SDRAM timing configuration 3	R/W	0x0000_0000	10.4.1.3/10-13
0x104	TIMING_CFG_0—DDR SDRAM timing configuration 0	R/W	0x0011_0105	10.4.1.4/10-14
0x108	TIMING_CFG_1—DDR SDRAM timing configuration 1	R/W	0x0000_0000	10.4.1.5/10-16
0x10C	TIMING_CFG_2—DDR SDRAM timing configuration 2	R/W	0x0000_0000	10.4.1.6/10-18
0x110	DDR_SDRAM_CFG—DDR SDRAM control configuration	R/W	0x0200_0000	10.4.1.7/10-19
0x114	DDR_SDRAM_CFG_2—DDR SDRAM control configuration 2	R/W	0x0000_0000	10.4.1.8/10-22
0x118	DDR_SDRAM_MODE—DDR SDRAM mode configuration	R/W	0x0000_0000	10.4.1.9/10-23
0x11C	DDR_SDRAM_MODE_2—DDR SDRAM mode configuration 2	R/W	0x0000_0000	10.4.1.10/10-24
0x120	DDR_SDRAM_MD_CNTL—DDR SDRAM mode control	R/W	0x0000_0000	10.4.1.11/10-25
0x124	DDR_SDRAM_INTERVAL—DDR SDRAM interval configuration	R/W	0x0000_0000	10.4.1.12/10-27
0x128	DDR_DATA_INIT—DDR SDRAM data initialization	R/W	0x0000_0000	10.4.1.13/10-28
0x130	DDR_SDRAM_CLK_CNTL—DDR SDRAM clock control	R/W	0x0200_0000	10.4.1.14/10-28

Table A-14. DDR Memory Controller Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x140–0x144	Reserved	—	—	—
0x148	DDR_INIT_ADDR—DDR training initialization address	R/W	0x0000_0000	10.4.1.15/10-29
0x150–0x	Reserved	—	—	—
0xBF8	DDR_IP_REV1—DDR IP block revision 1	R	0xn ¹ n ¹ n ¹ n ¹ _n ¹ n ¹ n ¹ n ¹	10.4.1.16/10-29
0xBF8	DDR_IP_REV2—DDR IP block revision 2	R	0x00nn_00nn	10.4.1.17/10-30
0xE00	DATA_ERR_INJECT_HI—Memory data path error injection mask high	R/W	0x0000_0000	10.4.1.18/10-30
0xE04	DATA_ERR_INJECT_LO—Memory data path error injection mask low	R/W	0x0000_0000	10.4.1.19/10-31
0xE08	ERR_INJECT—Memory data path error injection mask ECC	R/W	0x0000_0000	10.4.1.20/10-31
0xE20	CAPTURE_DATA_HI—Memory data path read capture high	R/W	0x0000_0000	10.4.1.21/10-32
0xE24	CAPTURE_DATA_LO—Memory data path read capture low	R/W	0x0000_0000	10.4.1.22/10-32
0xE28	CAPTURE_ECC—Memory data path read capture ECC	R/W	0x0000_0000	10.4.1.23/10-33
0xE40	ERR_DETECT—Memory error detect	w1c	0x0000_0000	10.4.1.24/10-33
0xE44	ERR_DISABLE—Memory error disable	R/W	0x0000_0000	10.4.1.25/10-34
0xE48	ERR_INT_EN—Memory error interrupt enable	R/W	0x0000_0000	10.4.1.26/10-35
0xE4C	CAPTURE_ATTRIBUTES—Memory error attributes capture	R/W	0x0000_0000	10.4.1.27/10-36
0xE50	CAPTURE_ADDRESS—Memory error address capture	R/W	0x0000_0000	10.4.1.28/10-37
0xE54	Reserved	—	—	—
0xE58	ERR_SBE—Single-Bit ECC memory error management	R/W	0x0000_0000	10.4.1.29/10-37

¹ Implementation-dependent reset values are listed in specified section/page.

A.15 I²C Controller

 Table A-15. I²C Memory Map

Address	I ² C Register	Access	Reset	Section/Page
I ² C Controller 1—Block Base Address 0x0_3000 I ² C Controller 2—Block Base Address 0x0_3100				
0x0_3000	I2C1ADR—I ² C1 address register	R/W	0x00	17.3.1.1/17-5
0x0_3004	I2C1FDR—I ² C1 frequency divider register	R/W	0x00	17.3.1.2/17-6
0x0_3008	I2C1CR—I ² C1 control register	R/W	0x00	17.3.1.3/17-7
0x0_300C	I2C1SR—I ² C1 status register	R/W	0x81	17.3.1.4/17-8
0x0_3010	I2C1DR—I ² C1 data register	R/W	0x00	17.3.1.5/17-9
0x0_3014	I2C1DFSRR—I ² C1 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—I ² C2 address register	R/W	0x00	17.3.1.1/17-5
0x0_3104	I2C2FDR—I ² C2 frequency divider register	R/W	0x00	17.3.1.2/17-6
0x0_3108	I2C2CR—I ² C2 control register	R/W	0x00	17.3.1.3/17-7
0x0_310C	I2C2SR—I ² C2 status register	R/W	0x81	17.3.1.4/17-8
0x0_3110	I2C2DR—I ² C2 data register	R/W	0x00	17.3.1.5/17-9
0x0_3114	I2C2DFSRR—I ² C2 digital filter sampling rate register	R/W	0x10	17.3.1.6/17-10
0x0_311C– 0x0_31FF	Reserved, should be cleared	—	—	—

A.16 DUART

Table A-16. DUART Registers

Offset	Register	Access	Reset	Section/Page
0x0_4500 0x0_4900–49 10	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	18.3.1.7/18-11
0x0_4503	ULCR—ULCR[DLAB] = x UART1 line control register	R/W	0x00	18.3.1.8/18-12
0x0_4504	UMCR—ULCR[DLAB] = x UART1 MODEM control register	R/W	0x00	18.3.1.9/18-14

Table A-16. DUART Registers (continued)

Offset	Register	Access	Reset	Section/Page
0x0_4505	ULSR—ULCR[DLAB] = x UART1 line status register	R	0x60	18.3.1.10/18-15
0x0_4506	UMSR—ULCR[DLAB] = x UART1 MODEM status register	R	0x00	18.3.1.11/18-16
0x0_4507	USCR—ULCR[DLAB] = x UART1 scratch register	R/W	0x00	18.3.1.12/18-17
0x0_4510	UDSR—ULCR[DLAB] = x UART1 DMA status register	R	0x01	18.3.1.13/18-17
0x0_4600 0x0_4A00–4A10	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	18.3.1.7/18-11
0x0_4603	ULCR—ULCR[DLAB] = x UART2 line control register	R/W	0x00	18.3.1.8/18-12
0x0_4604	UMCR—ULCR[DLAB] = x UART2 MODEM control register	R/W	0x00	18.3.1.9/18-14
0x0_4605	ULSR—ULCR[DLAB] = x UART2 line status register	R	0x60	18.3.1.10/18-15
0x0_4606	UMSR—ULCR[DLAB] = x UART2 MODEM status register	R	0x00	18.3.1.11/18-16
0x0_4607	USCR—ULCR[DLAB] = x UART2 scratch register	R/W	0x00	18.3.1.12/18-17
0x0_4610	UDSR—ULCR[DLAB] = x UART2 DMA status register	R	0x01	18.3.1.13/18-17

A.17 Enhanced Local Bus Controller (eLBC)

Table A-17. Enhanced Local Bus Controller Registers

Enhanced Local Bus Controller—Block Base Address				
Offset	Register	Access	Reset	Section/Page
0x000	BR0—Base register 0	R/W	0x0000_????	11.3.1.1/11-9
0x008	BR1—Base register 1	R/W	All zeros	11.3.1.1/11-9
0x010	BR2—Base register 2	R/W	All zeros	11.3.1.1/11-9
0x018	BR3—Base register 3	R/W	All zeros	11.3.1.1/11-9
0x020	BR4—Base register 4	R/W	All zeros	11.3.1.1/11-9
0x028	BR5—Base register 5	R/W	All zeros	11.3.1.1/11-9
0x030	BR6—Base register 6	R/W	All zeros	11.3.1.1/11-9
0x038	BR7—Base register 7	R/W	All zeros	11.3.1.1/11-9
0x004	OR0—Options register 0	R/W	0x0000_0FF7	11.3.1.2/11-11

Table A-17. Enhanced Local Bus Controller Registers (continued)

Enhanced Local Bus Controller—Block Base Address				
Offset	Register	Access	Reset	Section/Page
0x00C	OR1—Options register 1	R/W	All zeros	11.3.1.2/11-11
0x014	OR2—Options register 2	R/W	All zeros	11.3.1.2/11-11
0x01C	OR3—Options register 3	R/W	All zeros	11.3.1.2/11-11
0x024	OR4—Options register 4	R/W	All zeros	11.3.1.2/11-11
0x02C	OR5—Options register 5	R/W	All zeros	11.3.1.2/11-11
0x034	OR6—Options register 6	R/W	All zeros	11.3.1.2/11-11
0x03C	OR7—Options register 7	R/W	All zeros	11.3.1.2/11-11
0x068	MAR—UPM address register	R/W	All zeros	11.3.1.3/11-19
0x06C	Reserved	—	—	—
0x070	MAMR—UPMA mode register	R/W	All zeros	11.3.1.4/11-20
0x074	MBMR—UPMB mode register	R/W	All zeros	11.3.1.4/11-20
0x078	MCMR—UPMC mode register	R/W	All zeros	11.3.1.4/11-20
0x07C–0x080	Reserved	—	—	—
0x084	MRTPR—Memory refresh timer prescaler register	R/W	All zeros	11.3.1.5/11-22
0x088	MDR—UPM/FCM data register	R/W	All zeros	11.3.1.6/11-22
0x08C	Reserved	—	—	—
0x090	LSOR—Special operation initiation register	R/W	All zeros	11.3.1.7/11-23
0x094–0x09C	Reserved	—	—	—
0x0A0	LURT—UPM refresh timer	R/W	All zeros	11.3.1.8/11-24
0x0A4–0x0AC	Reserved	—	—	—
0x0B0	LTESR—Transfer error status register	w1c	All zeros	11.3.1.9/11-25
0x0B4	LTEDR—Transfer error disable register	R/W	All zeros	11.3.1.10/11-27
0x0B8	LTEIR—Transfer error interrupt register	R/W	All zeros	11.3.1.11/11-28
0x0BC	LTEATR—Transfer error attributes register	R/W	All zeros	11.3.1.12/11-29
0x0C0	LTEAR—Transfer error address register	R/W	All zeros	11.3.1.13/11-30
0x0C–0x0CC	Reserved	—	—	—
0x0D0	LBCR—Configuration register	R/W		11.3.1.14/11-30
0x0D4	LCRR—Clock ratio register	R/W		11.3.1.15/11-32
0x0D8–0x0DC	Reserved	—	—	—
0x0E0	FMR—Flash mode register	R/W	0x0000_0n00	11.3.1.16/11-33
0x0E4	FIR—Flash instruction register	R/W	All zeros	11.3.1.17/11-34

Table A-17. Enhanced Local Bus Controller Registers (continued)

Enhanced Local Bus Controller—Block Base Address				
Offset	Register	Access	Reset	Section/Page
0x0E8	FCR—Flash command register	R/W	All zeros	11.3.1.18/11-35
0x0EC	FBAR—Flash block address register	R/W	All zeros	11.3.1.19/11-36
0x0F0	FPAR—Flash page address register	R/W	All zeros	11.3.1.20/11-36
0x0F4	FBCR—Flash byte count register	R/W	All zeros	11.3.1.21/11-38
0x0F8–0x0FC	Reserved	—	—	—

A.18 Serial Peripheral Interface (SPI)

Table A-18. Serial Peripheral Interface (SPI) Registers

Serial Peripheral Interface (SPI)—Block Base Address 0x0_7000				
Offset	Register	Access	Reset	Section/Page
0x000–0x01F	Reserved	—	—	—
0x020	SPI mode register (SPMODE)	R/W	0x0000_0000	19.4.1.1/1919-9
0x024	SPI event register (SPIE)	Mixed	0x0000_0000	19.4.1.2/1919-12
0x028	SPI mask register (SPIM)	R/W	0x0000_0000	19.4.1.3/1919-13
0x02C	SPI command register (SPCOM)	W	0x0000_0000	19.4.1.4/1919-14
0x030	SPI transmit register (SPITD)	W	0x0000_0000	19.4.1.5/1919-14
0x034	SPI receive register (SPIRD)	R	0xFFFF_FFFF	19.4.1.6/1919-15
0x038–0xFFFF	Reserved	—	—	—

A.19 DMA Engine 1

Table A-19. DMA Engine 1

Offset	Register	Access	Reset	Section/Page
0x000	DMACR—DMA Control Register	R/W	0x0000_E400	13.2.1/13-4
0x004	DMAES—DMA Error Status Register	RO	0x0000_0000	13.3/13-5
0x008–0x00B	Reserved	—	—	—
0x00C	DMAERQ	*some*	*some*	*some*
0x00D–0x010	Reserved	—	—	—
0x014	DMAEEI—DMA enable error interrupt register	R/W	0x0000_0000	13.3.2/13-8
0x018	DMASERQ—DMA set enable request	R/W	0x0000_0000	13.3.3/13-9
0x019	DMACERQ—DMA clear enable request	R/W	0x0000_0000	13.3.3/13-9

Table A-19. DMA Engine 1

Offset	Register	Access	Reset	Section/Page
0x01A	DMASEEI—DMA Set Enable Error Interrupt	R/W	0x0000_0000	13.3.3/13-9
0x01B	DMACEEI—DMA Clear Enable Error Interrupt	R/W	0x0000_0000	13.3.4/13-10
0x01C	DMACINT—DMA Clear Interrupt Request	R/W	0x0000_0000	13.3.5/13-10
0x01D	DMACERR—DMA Clear Error	R/W	0x0000_0000	13.3.6/13-11
0x01E	DMASSRT—DMA Set START Bit	R/W	0x0000_0000	13.3.7/13-12
0x01F	DMACDNE—DMA Clear DONE Status Bit	R/W	0x0000_0000	13.3.8/13-12
0x020	Reserved	—	—	—
0x024	DMAINT—DMA interrupt request register	w1c	0x0000_0000	13.3.9/13-13
0x028	Reserved	—	—	—
0x02C	DMAERR—DMA error register	w1c	0x0000_0000	13.3.10/13-14
0x030– 0x034	Reserved	—	—	—
0x038	DMAGPOR—DMA general purpose output register	R/W	0x0000_0000	13.3.11/13-15
0x03C– 0x0FC	Reserved	—	—	—
0x100– 0x13C	DCHPRI n —DMA Channel n Priority Register	R/W	0x0000_nnnn	13.3.12/13-16
0x140– 0xFFC	Reserved	—	—	—

A.20 DMA Engine 2

Table A-20. Module Memory Map

Offset	Register	Access	Reset	Section/Page
0x0_8100	DMAMR0—DMA 0 mode register	R/W	All zeros	14.3.8.1/14-9
0x0_8104	DMASR0—DMA 0 status register	R/W	All zeros	14.3.8.2/14-11
0x0_8108	DMACDAR0—DMA 0 current descriptor address register	R/W	All zeros	14.3.8.3/14-12
0x0_8110	DMASAR0—DMA 0 source address register	R/W	All zeros	14.3.8.4/14-13
0x0_8118	DMADAR0—DMA 0 destination address register	R/W	All zeros	14.3.8.5/14-13
0x0_8120	DMABCR0—DMA 0 byte count register	R/W	All zeros	14.3.8.6/14-14
0x0_8124	DMANDAR0—DMA 0 next descriptor address register	R/W	All zeros	14.3.8.7/14-14
0x0_8180	DMAMR1—DMA 1 mode register	R/W	All zeros	14.3.8.1/14-9
0x0_8184	DMASR1—DMA 1 status register	R/W	All zeros	14.3.8.2/14-11
0x0_8188	DMACDAR1—DMA 1 current descriptor address register	R/W	All zeros	14.3.8.3/14-12

Table A-20. Module Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x0_8190	DMASAR1—DMA 1 source address register	R/W	All zeros	14.3.8.4/14-13
0x0_8198	DMADAR1—DMA 1 destination address register	R/W	All zeros	14.3.8.5/14-13
0x0_81A0	DMABCR1—DMA 1 byte count register	R/W	All zeros	14.3.8.6/14-14
0x0_81A4	DMANDAR1—DMA 1 next descriptor address register	R/W	All zeros	14.3.8.7/14-14
0x0_8200	DMAMR2—DMA 2 mode register	R/W	All zeros	14.3.8.1/14-9
0x0_8204	DMASR2—DMA 2 status register	R/W	All zeros	14.3.8.2/14-11
0x0_8208	DMACDAR2—DMA 2 current descriptor address register	R/W	All zeros	14.3.8.3/14-12
0x0_8210	DMASAR2—DMA 2 source address register	R/W	All zeros	14.3.8.4/14-13
0x0_8218	DMADAR2—DMA 2 destination address register	R/W	All zeros	14.3.8.5/14-13
0x0_8220	DMABCR2—DMA 2 byte count register	R/W	All zeros	14.3.8.6/14-14
0x0_8224	DMANDAR2—DMA 2 next descriptor address register	R/W	All zeros	14.3.8.7/14-14
0x0_8280	DMAMR3—DMA 3 mode register	R/W	All zeros	14.3.8.1/14-9
0x0_8284	DMASR3—DMA 3 status register	R/W	All zeros	14.3.8.2/14-11
0x0_8288	DMACDAR3—DMA 3 current descriptor address register	R/W	All zeros	14.3.8.3/14-12
0x0_8290	DMASAR3—DMA 3 source address register	R/W	All zeros	14.3.8.4/14-13
0x0_8298	DMADAR3—DMA 3 destination address register	R/W	All zeros	14.3.8.5/14-13
0x0_82A0	DMABCR3—DMA 3 byte count register	R/W	All zeros	14.3.8.6/14-14
0x0_82A4	DMANDAR3—DMA 3 next descriptor address register	R/W	All zeros	14.3.8.7/14-14
0x0_82A8	DMAGSR—DMA general status register	R	All zeros	14.3.8.8/14-15
0x0_82B0– 0x0_82FF	Reserved	—	—	—

A.21 Enhanced Secure Digital Host Controller (eSDHC)

Table A-21. Enhanced Secure Digital Host Controller (eSDHC) Registers

	Register	Access	Reset	Section/Page
0x000	DMA system address (DSADDR)	R/W	0x0000_0000	12.4.1/12-7
0x004	Block attributes (BLKATTR)	R/W	0x0001_0000	12.4.2/12-7
0x008	Command argument (CMDARG)	R/W	0x0000_0000	12.4.3/12-8
0x00C	Command transfer type (XFERTYP)	R/W	0x0000_0000	12.4.4/12-9
0x010	Command response0 (CMDRSP0)	R	0x0000_0000	12.4.5/12-12
0x014	Command response1 (CMDRSP1)	R	0x0000_0000	12.4.5/12-12
0x018	Command response2 (CMDRSP2)	R	0x0000_0000	12.4.5/12-12
0x01C	Command response3 (CMDRSP3)	R	0x0000_0000	12.4.5/12-12

Table A-21. Enhanced Secure Digital Host Controller (eSDHC) Registers (continued)

	Register	Access	Reset	Section/Page
0x020	Data buffer access port (DATPORT)	R/W	0x0000_0000	12.4.6/12-14
0x024	Present state (PRSSTAT)	R	0x0F80_00F8	12.4.7/12-15
0x028	Protocol control (PROCTL)	R/W	0x0000_0020	12.4.8/12-19
0x02C	System control (SYSCTL)	Mixed	0x0000_8008	12.4.9/12-22
0x030	Interrupt status (IRQSTAT)	w1c	0x0000_0000	12.4.10/12-24
0x034	Interrupt status enable (IRQSTATEN)	R/W	0x117F_013F	12.4.11/12-29
0x038	Interrupt signal enable (IRQSIGEN)	R/W	0x0000_0000	12.4.12/12-31
0x03C	Auto CMD12 status (AUTO12ERR)	R	0x0000_0000	12.4.13/12-33
0x040	Host controller capabilities (HOSTCAPBLT)	R	0x01F3_0000	12.4.14/12-35
0x044 ¹	Watermark level (WML)	R/W	0x1010_1010	12.4.15/12-36
0x050	Force event (FEVT)	W	0x0000_0000	12.4.16/12-37
0x0FC	Host controller version (HOSTVER)	R	0x0000_1201	12.4.17/12-39
0x40C	DMA control register (DCR)	R/W	0x0000_0000	12.4.18/12-39

¹ The addresses following 0x044, except 0x050, 0x0FC and 0x40C, are reserved and read as all 0s. Writes to these registers are ignored.

A.22 FlexCAN

Table A-22. FlexCAN Memory Map

Offset	Register	Access	Reset	Affected by Hard Reset	Affected by Soft Reset	Section/Page
CAN1—Block Base Address CAN2—Block Base Address						
0x000	Module configuration (MCR)	S		Yes	Yes	15.3.3.1/15-12
0x004	Control register (CTRL)	S/U	All zeros	Yes	No	15.3.3.2/15-15
0x008	Free running timer (TIMER)	S/U	All zeros	Yes	Yes	15.3.3.3/15-18
0x00C	Reserved		—	—	—	—
0x010	Rx Global Mask (RXGMASK)	S/U	FFFF_FFFF	Yes	No	15.3.3.4/15-19
0x014	Rx Buffer 14 Mask (RX14MASK)	S/U	FFFF_FFFF	Yes	No	15.3.3.5/15-19
0x018	Rx Buffer 15 Mask (RX15MASK)	S/U	FFFF_FFFF	Yes	No	15.3.3.6/15-20
0x01C	Error Counter Register (ECR)	S/U	All zeros	Yes	Yes	15.3.3.7/15-20
0x020	Error and Status Register (ESR)	S/U	All zeros	Yes	Yes	15.3.3.8/15-22

Table A-22. FlexCAN Memory Map (continued)

Offset	Register	Access	Reset	Affected by Hard Reset	Affected by Soft Reset	Section/ Page
CAN1—Block Base Address CAN2—Block Base Address						
0x024	Interrupt Masks 2 (IMASK2)	S/U	All zeros	Yes	Yes	15.3.3.9/15-24
0x028	Interrupt Masks 1 (IMASK1)	S/U	All zeros	Yes	Yes	15.3.3.10/15-25
0x02C	Interrupt Flags 2 (IFLAG2)	S/U	All zeros	Yes	Yes	15.3.3.11/15-25
0x030	Interrupt Flags 1 (IFLAG1)	S/U	All zeros	Yes	Yes	15.3.3.12/15-26
0x034– 0x05F	Reserved	—	—	—	—	—
0x060– 0x07F	Reserved		—			—
0x080– 0x17F	Message Buffers MB0–MB15	S/U		No	No	15.3.1/15-7
0x180– 0x27F	Message Buffers MB16–MB31	S/U		No	No	15.3.1/15-7
0x280– 0x047F	Message Buffers MB32–MB63	S/U		No	No	15.3.1/15-7
0x480– 087F	Reserved	—	—	—	—	—
0x880– 0x08BF	Rx Individual Mask Registers RXIMR0–RXIMR15	S/U		No	No	15.3.3.13/15-27
0x8C0– 0x08FF	Rx Individual Mask Registers RXIMR16–RXIMR31	S/U		No	No	15.3.3.13/15-27
0x900– 0x097F	Rx Individual Mask Registers RXIMR32–RXIMR63	S/U		No	No	15.3.3.13/15-27

A.23 PCI Configuration Access Registers

Table A-23. PCI Configuration Access Registers

Offset	Register	Access	Reset	Section/Page
PCI Configuration Access Registers—Block Base Address 0x0_8300				
0x00	PCI_CONFIG_ADDRESS	W	All zeros	23.3.1.1/23-12
0x04	PCI_CONFIG_DATA	R/W	All zeros	23.3.1.2/23-14
0x08	PCI_INT_ACK	R	N/A	23.3.1.3/23-15

A.24 PCI Memory Mapped Registers

Table A-24. PCI Memory-Mapped Registers

Offset	Register	Access	Reset	Section/Page
PCI Controller—Block Base Address 0x0_8500				
PCI Error Management Registers				
0x00	PCI error status register (PCI_ESR)	w1c	All zeros	23.3.2.1/23-15
0x04	PCI error capture disable register (PCI_ECDR)	R/W	All zeros	23.3.2.2/23-16
0x08	PCI error enable register (PCI_EER)	R/W	All zeros	23.3.2.3/23-17
0x0C	PCI error attributes capture register (PCI_EATCR)	R/W	All zeros	23.3.2.4/23-18
0x10	PCI error address capture register (PCI_EACR)	R	All zeros	23.3.2.5/23-19
0x14	PCI error extended address capture register (PCI_EEACR)	R	All zeros	23.3.2.6/23-20
0x18	PCI error data capture register (PCI_EDCR)	R/W	All zeros	23.3.2.7/23-20
PCI Control and Status Registers				
0x20	PCI general control register (PCI_GCR)	R/W	All zeros	23.3.2.8/23-20
0x24	PCI error control register (PCI_ECR)	R/W	All zeros	23.3.2.9/23-21
0x28	PCI general status register (PCI_GSR)	R	All zeros	23.3.2.10/23-22
PCI Inbound ATU Registers				
0x38	PCI inbound translation address register 2 (PITAR2)	R/W	All zeros	23.3.2.11/23-22
0x3C	Reserved	—	—	—
0x40	PCI inbound base address register 2 (PIBAR2)	R/W	All zeros	23.3.2.12/23-23
0x44	PCI inbound extended base address register 2 (PIEBAR2)	R/W	All zeros	23.3.2.13/23-23
0x48	PCI inbound window attributes register 2 (PIWAR2)	R/W	All zeros	23.3.2.14/23-24
0x50	PCI inbound translation address register 1 (PITAR1)	R/W	All zeros	23.3.2.11/23-22
0x54	Reserved	—	—	—
0x58	PCI inbound base address register 1 (PIBAR1)	R/W	All zeros	23.3.2.12/23-23
0x5C	PCI inbound extended base address register 1 (PIEBAR1)	R/W	All zeros	23.3.2.13/23-23
0x60	PCI inbound window attributes register 1 (PIWAR1)	R/W	All zeros	23.3.2.14/23-24
0x68	PCI inbound translation address register 0 (PITAR0)	R/W	All zeros	23.3.2.11/23-22
0x6C	Reserved	—	—	—
0x70	PCI inbound base address register 0 (PIBAR0)	R/W	All zeros	23.3.2.12/23-23
0x74	Reserved	—	—	—
0x78	PCI inbound window attributes register 0 (PIWAR0)	R/W	All zeros	23.3.2.13/23-23
0x7C–0xFF	Reserved	—	—	—

A.25 Universal Serial Bus (USB) Interface

Table A-25. USB Interface Registers

Offset	Register	Access	Reset	Section/Page
USB DR Controller Registers				
0x000–0x0FF	Reserved, should be cleared	—	—	—
0x100	CAPLENGTH—Capability register length	R	0x40	16.3.1.1/1616-6
0x102	HCVERSION—Host interface version number	R	0x0100	16.3.1.2/1616-7
0x104	HCSPARAMS—Host controller structural parameters	R		16.3.1.3/1616-7
0x108	HCCPARAMS—Host controller capability parameters	R	0x0000_0006	16.3.1.4/1616-8
0x120	DCVERSION—Device interface version number	R	0x0001	16.3.1.5/1616-9
0x124	DCCPARAMS—Device controller parameters	R		16.3.1.6/1616-9
0x140	USBCMD—USB command	Mixed		16.3.2.1/1616-10
0x144	USBSTS—USB status	Mixed	0x0000_0000	16.3.2.2/1616-13
0x148	USBINTR—USB interrupt enable	R/W	0x0000_0000	16.3.2.3/1616-15
0x14C	FRINDEX—USB frame index	R/W	All zeros	16.3.2.4/1616-16
0x154	PERIODICLISTBASE—Frame list base address ¹	R/W	All zeros	16.3.2.6/1616-18
	DEVICEADDR—USB device address	R/W	0x0000_0000	16.3.2.7/1616-18
0x158	ASYNCLISTADDR—Next asynchronous list addr (host mode) ¹	R/W	0x0000_0000	16.3.2.8/1616-19
	ENDPOINTLISTADDR—Address at endpoint list (device mode)	R/W	0x0000_0000	16.3.2.9/1616-20
0x160	BURSTSIZE—Programmable burst size	R/W	0x0000_1010	16.3.2.10/1616-20
0x164	TXFILLTUNING—Host TT transmit pre-buffer packet tuning	R/W		16.3.2.11/1616-21
0x170	ULPI VIEWPORT—ULPI Register Access	Mixed	0x0000_0000	16.3.2.12/1616-22
0x180	CONFIGFLAG—Configured flag register	R	0x0000_0001	16.3.2.13/1616-24
0x184	PORTSC—Port status/control	Mixed		16.3.2.14/1616-24
0x1A4				
0x1A8	USBMODE—USB device mode	R/W	0x0000_0000	16.3.2.16/1616-32
0x1AC	ENDPTSETUPSTAT—Endpoint setup status	w1c	0x0000_0000	16.3.2.17/1616-33
0x1B0	ENDPOINTPRIME—Endpoint initialization	R/W	0x0000_0000	16.3.2.18/1616-33
0x1B4	ENDPTFLUSH—Endpoint de-initialize	R/W	0x0000_0000	16.3.2.19/1616-34
0x1B8	ENDPTSTATUS—Endpoint status	R	0x0000_0000	16.3.2.20/1616-35
0x1BC	ENDPTCOMPLETE—Endpoint complete	w1c	0x0000_0000	16.3.2.21/1616-35
0x1C0	ENDPTCTRL0—Endpoint control 0	Mixed	0x0080_0080	16.3.2.22/1616-36
0x1C4	ENDPTCTRL1—Endpoint control 1	R/W	0x0000_0000	16.3.2.23/1616-37
0x1C8	ENDPTCTRL2—Endpoint control 2	R/W	0x0000_0000	16.3.2.23/1616-37

Table A-25. USB Interface Registers (continued)

Offset	Register	Access	Reset	Section/Page
0x400	SNOOP1—Snoop 1	R/W	0x0000_0000	16.3.2.24/1616-39
0x404	SNOOP2—Snoop 2	R/W	0x0000_0000	16.3.2.24/1616-39
0x408	AGE_CNT_THRESH—Age count threshold	R/W	0x0000_0000	16.3.2.25/1616-40
0x40C	PRI_CTRL—Priority control	R/W	0x0000_0000	16.3.2.26/1616-41
0x410	SI_CTRL—System interface control	R/W	0x0000_0000	16.3.2.27/1616-42
0x500	CONTROL—Control		0x0000_0000	16.3.2.28/1616-43
0x504– 0xFFF	Reserved, should be cleared	—	—	—

¹ This register has separate functions for the host and device operation; the host function is listed first in the table.



Appendix B

Revision History

B.1 Changes From Revision 1 to Revision 2

This appendix provides a list of major differences between revision 1 to 2 of the *MPC8309 PowerQUICC II Pro Integrated Communications Processor Reference Manual*.

Table B-1. Changes from Revision 1 to Revision 2

Signal Descriptions	
Table 3-1, MPC8309 Signal Reference by Functional Block , Figure 3-2, MPC8309 Signal Groupings (2 of 3)	Removed the following signals: FEC1_TMR_TX_ESFD FEC1_TMR_RX_ESFD FEC2_TMR_TX_ESFD FEC2_TMR_RX_ESFD
Reset, Clocking, and Initialization	
Section 4.4, "Clocking"	Updated Figure 4-7, "MPC8309 Clock Subsystem" .
Section 4.3.3.3, "Default Reset Configuration Words"	Updated the description given in the Meaning column for COREDIS in Table 4-21, "Hard-Coded Reset Configuration Word High Field Values" .
Section 4.3.2.2, "Reset Configuration Word High Register (RCWHR)"	Added reset value in Figure 4-4 "Reset Configuration Word High Register (RCWHR)" .
Section 4.3.2.1, "Reset Configuration Word Low Register (RCWLR)"	Added reset value in Figure 4-3 "Reset Configuration Word Low Register (RCWLR)" .
Table 4-23, Reset Configuration and Status Registers Memory Map	Updated reset values for RCWLR and RCWHR.
Table 4-20, Hard Coded Reset Configuration Word Low Fields Values	Updated the table.
System Boot	
Section 5.3.3, "EEPROM Data Structure"	In the second row of Table 5-5, "SPI EEPROM Data Structure," updated the location value specified in the second sentence of the Data bits column from "0x424F0_4F54" to "0x424F_4F54".
System Configuration	
Table 6-28, SICR_2 Bit Settings for MPC8309	Removed the following signals: FEC1_TMR_TX_ESFD FEC1_TMR_RX_ESFD FEC2_TMR_TX_ESFD FEC2_TMR_RX_ESFD

Table B-1. Changes from Revision 1 to Revision 2

Table 6-20, “System Configuration Register Memory Map” Section 6.3.2.9, “DDR Control Driver Register (DDRCDR)”	Updated the reset value of DDRCDR register as “All Zeros” in Figure 6-16, “DDR Control Driver Register (DDRCDR)” .
Section 6.7.5, “GTM Memory Map/Register Definition	Updated the Table 6-61, “GTM Register Address Map,” to add the General Purpose (Global) Timer Module 2-Block Base Address 0x0_0600. Updated the table subheading from: “General Purpose (Global) Timer Module 1-Block Base Address 0x0_0500” to: “General Purpose (Global) Timer Module 1-Block Base Address 0x0_0500 General Purpose (Global) Timer Module 2-Block Base Address 0x0_0600” Also, added a note at the end of the table to indicate that the offset will be same for the registers of Global Timer Module 2 timers (GTM5-8).
Section 6.3.2.6, “System I/O Configuration Register 2 (SICR_2)”	Added Table 6-29, “QE UART Multiplexing Details” to provide details on QE UART multiplexing.
Section 6.3.2.6, “System I/O Configuration Register 2 (SICR_2)”	In Table 6-28, “SICR_2 Bit Settings for MPC8309,” added table footnote for the pins multiplexed with QE.
Section 6.3.2.5, “System I/O Configuration Register 1 (SICR_1)”	In Table 6-27, “SICR_1 Bit Settings for MPC8309,” added table footnote for the pins multiplexed with QE.
Integrated Programmable Interrupt Controller (IPIC)	
Section 9.5.13, “System External Interrupt Mask Register (SEMSR)” (Figure 9-16)	In Figure 9-16, System External Interrupt Mask Register (SEMSR) , changed the following text appearing under the reset values of 0–15 bits from: “The reset values of implemented bits reflect the values of the external IRQ signals. Reserved bits are zeros.” to “All zeros”. In addition, removed the following second footnote for all 83xx devices that says: “The user should drive all IRQ inputs to an inactive state prior to reset negation”
Section 9.1, “Introduction”	Updated the features list to indicate the support of two global timer blocks.
<ul style="list-style-type: none"> • Figure 9-1, “Interrupt Sources Block Diagram for MPC8309” • Figure 9-30, “Interrupt Structure for MPC8309” 	Updated the figures to show 8 GTMs.
Section 9.6.6, “Interrupt Source Priorities”	In Table 9-38, “Interrupt Source Priority Levels,” updated the Interrupt Source Description column for Priorities- 62, 80, 98, and 116.
Enhanced Local Bus Controller	
Throughout	Updated LBIUCM to LBCM.
Throughout	Updated the complete chapter to support four chip selects.
Section 11.3.1.21, “Flash Byte Count Register (FBCR)”	Added the reset value in Figure 11-26, “Flash Byte Count Register” .

Table B-1. Changes from Revision 1 to Revision 2

Section 11.4.1.5, "Bus Monitor"	In the first sentence of the first paragraph modified 'each bus cycle' to 'each transaction'. Last sentence of note updated as follows: "To avoid such cases, it is recommended that while using FCM the bus monitor timeout be programmed to its maximum setting of LBCR[BMT] = 0 and LBCR[BMTPS] = 0xF".
Enhanced Secure Digital Host Controller	
Section 12.4.10, "Interrupt Status Register (IRQSTAT)"	In Table 12-14, "IRQSTAT Field Descriptions," appended the following sentence (after the second sentence) in the IRQSTAT[TC] bit field description: BLKATTR[BLKCNT] will be decremented when IRQSTAT[TC] is set.
Section 12.4.3, "Command Argument Register (CMDARG)"	In Table 12-5, "CMDARG Field Descriptions," updated "If PRSSTAT[CMD] is set, this register is write-protected." to "If PRSSTAT[CIHB] is set, this register is write-protected".
Section 12.4.10, "Interrupt Status Register (IRQSTAT)"	In Table 12-14, "IRQSTAT Field Descriptions," corrected bit field description of CIE (bit 12) from: 1 Timeout to: 1 Error.
Section 12.4.9, "System Control Register (SYSCTL)"	In Table 12-13, "SYSCTL Field Descriptions," updated last bulleted item of IPGEN (bit 31) field description to say the following: "The internal bus clock is not gated off".
Section 12.4.16, "Force Event Register (FEVT)"	In Table 12-24, "FEVT Field Descriptions," corrected the bit name of bit 15 from "FEVTCCE" to "FEVCTOE".
Section 12.1, "Overview"	Updated Figure 12-2, "eSDHC Block Diagram," as follows: 1) The "eSDHC controller clock" arrow is now connected to the "Clock controller & divider" block; 2) An arrow is added from the "Clock controller & divider" block to the "Clock & reset manager" block; 3) The dangling double arrow at the lower left corner of the "Data channel state machine" block is removed.
Section 12.4.10, "Interrupt Status Register (IRQSTAT)"	In Table 12-14, "IRQSTAT Field Descriptions," added the following note to the bit descriptions for IRQSTAT[CIE] and IRQSTAT[CCE]: Note: Under rare conditions, CIE/CCE may be set for a command without data while a command with data is in progress. On detecting a command index error, software should perform error recovery and reissue both the command with data and the command without data.
Section 12.4.4, "Transfer Type Register (XFERTYP)"	In the Table 12-8, "Relation Between Parameters and Name of Response Type," updated the fourth row to add R7 as the response type.
Section 12.6.5, "Commands for MMC/SD/SDIO"	- Added CMD8 row for SD Cards in Table 12-26, "Commands for MMC/SD/SDIO" - Added following note: <ul style="list-style-type: none"> • CMD8 differs for MMC and SD cards. For SD cards, CMD8 is referred to as SEND_IF_COND. For MMC cards, CMD8 is referred to as SEND_EXT_CSD.

Table B-1. Changes from Revision 1 to Revision 2

<p>Section 12.5.5, “Clock Generator”</p>	<p>Removed the word “base” from the following sentence: The frequency of clock output from this stage, DIV, can be base, base/2, base/4, ..., or base/256. The sentence now reads as follows: The frequency of clock output from this stage, DIV, can be base/2, base/4, ..., or base/256.</p>
<p>Universal Serial Bus Interface</p>	
<p>Section 16.5.1, “Periodic Frame List”</p>	<ul style="list-style-type: none"> • Removed the following sentence occurring after Figure 16-35, “Periodic Schedule Organization”: “Split transaction interrupt, bulk and control are also managed using queue heads and queue element transfer descriptors.” • Updated the third paragraph starting from “The periodic frame list is...” as follows: “The periodic frame list is a 4K-page aligned array of Frame List Link pointers. The length of the frame list is programmable. The programmability of the periodic frame list is exported to system software through the HCCPARAMS register. The length can be selected by system software as one of 8, 16, 32, 64, 128, 256, 512 or 1024 elements. 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.”
<p>Section 16.6.1, “Host Controller Initialization”</p>	<p>Removed the following sentence in the first paragraph: “After a hardware reset, only the operational registers are at their default values.”</p>
<p>Section 16.6.2, “Power Port”</p>	<p>Added the following sentence at the end of the paragraph: “The Configured Flag and Port Power Control bits are always 1 in Host Mode. The PPE always follows the state of Port Power (PP) bit that is, if PP is 0, PPE will be 0 and if PP is 1, PPE will be 1.”</p>
<p>Section 16.6.4.1, “Port Suspend/Resume”</p>	<p>In Table 16-63, “Behavior During Wake-Up Events,” updated the following bit names in “Port Status and Signaling Type” column:</p> <ul style="list-style-type: none"> • WKDSCNNT_E → PORTSC[WKDS] • WKCNT_E → PORTSC[WKCN] • WKOC_E → PORTSC[WKOC]
<p>Section 16.8.3.2.2, “Data Toggle Inhibit”</p>	<p>Updated “data toggle” to say as “data toggle state bit” in the first sentence of the second paragraph. The sentence now reads: “In normal operation, the USB_DR checks the DATA0/DATA1 bit against the data toggle state bit to determine if the packet is valid.”</p>
<p>Section 16.6.12.2.2, “Host Controller Operational Model for FSTNs”</p>	<p>In Figure 16-55, “Example Host Controller Traversal of Recovery Path via FSTNs,” changed the notations of the nodes as per the corresponding diagram in “Enhanced Host Controller Interface (EHCI) Specification for Universal Serial Bus, Revision 1.0”.</p>
<p>Section 16.6.12.2.1, “Split Transaction Scheduling Mechanisms for Interrupt”</p>	<p>In Figure 16-54, “General Structure of EHCI Periodic Schedule Utilizing Interrupt Spreading,” changed the notations as per the corresponding diagram in “Enhanced Host Controller Interface (EHCI) Specification for Universal Serial Bus, Revision 1.0”.</p>
<p>Section 16.3.2.4, “Frame Index Register (FRINDEX)”</p>	<p>In Figure 16-11, “USB Frame Index (FRINDEX),” and Table 16-2, “USB Interface Memory Map,” updated reset value from “0x0000_nnnn” to “All zeros”.</p>
<p>Section, “Timing Diagrams”</p>	<p>Removed section.</p>

Table B-1. Changes from Revision 1 to Revision 2

Section 16.6.8.1, “Host Controller Operational Model for iTDs”	Updated the sentence beginning from “Note, that the host controller is not required to update....” in the second list bullet, seventh paragraph, to “Note, that the host controller does not update....”.
Section 16.3.2.6, “Periodic Frame List Base Address Register (PERIODICLISTBASE)”	In Figure 16-12, “Periodic Frame List Base Address (PERIODICLISTBASE),” and Table 16-2, “USB Interface Memory Map” changed the reset values from “0xnxxx_0000” to “all zeros”.
Section 16.2, “External Signals”	Removed the details starting from the following sentence: “Many of the signals for the PHY interfaces are muxed....” and its referring table.
Section 16.3.2.14, “Port Status and Control Register (PORTSC)”	Updated the name of bit 20 changed from WLCN to WKCN in Figure 16-20, “Port Status and Control (PORTSC)” and Table 16-22, “PORTSC Register Field Descriptions”.
Section 16.8.4, “Managing Queue Heads”	In Figure 16-64, “Endpoint Queue Head Diagram,” updated “Up to 32 Elements” to “Up to 6 Elements”.
Section 16.8.5.3, “Executing a Transfer Descriptor”	Updated the numbering for case1 and case2 lists. Updated the steps (7) and (8) listed under “Link list is not empty” case as follows: 7) If status bit read in (4) is ‘1’ DONE. 8) If status bit read in (4) is ‘0’ then Goto Case 1: Step 1.
Section 16.6.13, “Port Test Modes”	Updated the third step to remove the sentences starting from “Note that an EHCI host controller RS cleared and HCH set.” and append the following sentence: “In Device mode, the Test Mode starts only if Run/Stop bit is set to 1. In Host mode, the Test Mode starts regardless of Run/Stop bit.”
Throughout	Updated “USB_DR” to “USB_DR controller”.
Section 16.8.3.3, “Device Operational Model For Packet Transfers”	Updated the heading name from “Device For Packet Transfers” to “Device Operational Model For Packet Transfers”.
Throughout	Updated all instances of FTSN to FSTN.
Section 16.8.3.3.1, “Priming Transmit Endpoints”	Updated the following sentence in the second paragraph from: “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.” to “This FIFO is split into virtual channels so that the leading data can be stored for any endpoint”.
Section 16.6.12.3.3, “Split Transaction Execution State Machine for Isochronous”	Updated Figure 16-59, “Split Transaction State Machine for Isochronous,” as per the corresponding figure given in “Enhanced Host Controller Interface (EHCI) Specification for Universal Serial Bus, Revision 1.0”.
Section 16.6.12.2.5, “Split Transaction Execution State Machine for Interrupt”	Updated Figure 16-56, “Split Transaction State Machine for Interrupt,” as per the corresponding figure given in “Enhanced Host Controller Interface (EHCI) Specification for Universal Serial Bus, Revision 1.0”.

Table B-1. Changes from Revision 1 to Revision 2

<p>Section 16.6.12.3.4, “Periodic Isochronous—Do-Start-Split,”</p>	<p>Updated the following sentence in the seventh paragraph: “The preferred method is to detect when T-Count decrements to zero as a result of a start-split bus transaction. Equivalently, the host controller can detect when Total Bytes to Transfer decrements to zero. Either implementation must ensure that if the initial condition is Total Bytes to Transfer is equal to zero and T-count is equal to a one, the host controller issues a single start-split, with a zero-length data payload.” to: “Setting the Active bit to zero depends on siTD[TP] being 00 or 11, and siTD[Total Bytes to Transfer] decrements to 0.”</p>
<p>I2C Interfaces</p>	
<p>Section 17.4.5, “Boot Sequencer Mode”</p>	<p>Updated this section to adding step-by-step sequence for standard I2C addressing mode and extension of standard I2C addressing mode.</p>
<p>QUICC Engine Block on the MPC8309</p>	
<p>Section 24.2.2.7, “QUICC Engine Block Control—External Requests Device Specific Information”</p>	<p>Updated the QE UART multiplexing information in the last list item as follows:</p> <ul style="list-style-type: none"> - For the UART available on UCC1, UCC2 and UCC3 - CTS_B is available on RX_DV - CD_B is available on RX_ER - SIN is available on RXD0 - RTS_B is available on TX_EN - SOUT is available on TXD0

B.2 Changes From Revision 0 to Revision 1

This appendix provides a list of major differences between revision 0 to 1 of the *MPC8309 PowerQUICC II Pro Integrated Communications Processor Reference Manual*.

Table B-2. Changes from Revision 0 to Revision 1

Overview	
Section 1.1, “MPC8309 PowerQUICC II Pro Processor Overview”	In the first paragraph: Change From: “four FlexCAN interfaces with 16 message buffers each” Change To: “four FlexCAN interfaces with maximum 64 message buffers each”
Section 1.3.13, “DMA Engine 2”	Removed the following bullet: “Optional external control signals (REQ/ACK/DONE) per channel”
Signal Descriptions	
Table 3-1, MPC8309 Signal Reference by Functional Block	Made the following modifications in the table: <ul style="list-style-type: none"> • Changed the signal names HDLC2_RXCLK (CLK14) to HDLC2_RXCLK (CLK13) and HDLC2_TXCLK (CLK13) to HDLC2_TXCLK (CLK14) • For the signal QE_BRG[7], modified the Alternate Function(s) detail to HDLC2_TXCLK(CLK14)/GPIO[16]/TDM2_TCK(CLK6) • For the signal QE_BRG[8], modified the Alternate Function(s) detail to HDLC2_RXCLK(CLK13)/GPIO[17]/TDM2_RCK(CLK5) • For the signal TDM2_RCK (CLK5), modified the Alternate Function(s) detail to HDLC2_RXCLK(CLK13)/GPIO[17]/QE_BRG[8] • For the signal TDM2_TCK (CLK6), modified the Alternate Function(s) detail to HDLC2_TXCLK(CLK14)/GPIO[16]/QE_BRG[7]
Table 3-1, MPC8309 Signal Reference by Functional Block	Updated the following for signal FEC1_TMR_TX_ESFD (the one above FEC2_TMR_RX_ESFD): <ul style="list-style-type: none"> • Updated the signal name as FEC2_TMR_TX_ESFD • Updated the signal description as Transmit external start of frame delimiter (for FEC2) • Updated the alternate function(s) as FEC3_RXD[0] Updated the following for signal FEC2_TMR_RX_ESFD: <ul style="list-style-type: none"> • Updated the signal description as Receive external start of frame delimiter (for FEC2) • Updated the alternate function(s) as FEC3_RXD[1]
Figure 3-1, MPC8309 Signal Groupings (1 of 3) Figure 3-3, MPC8309 Signal Groupings (3 of 3) Table 3-1, MPC8309 Signal Reference by Functional Block	Added the signals QE_EXT_REQ_3 and QE_EXT_REQ_4.

Table B-2. Changes from Revision 0 to Revision 1

Figure 3-1, MPC8309 Signal Groupings (1 of 3) Table 3-1, MPC8309 Signal Reference by Functional Block	Changed signal name from $\overline{\text{SPI_CS}}$ to $\overline{\text{SPISEL_BOOT}}$.
Reset, Clocking, and Initialization	
Table 4-7, RCWLR Bit Settings	Updated the description of the bit CEVCOD(24–25) as follows: QUICC Engine PLL VCO division. Establishes the internal ratio between the QUICC Engine PLL VCO point and the QUICC Engine PLL output. 00 2 01 4 10 8 11 Reserved Notes: Set CEVCOD to 00(Division factor of 2) for QE frequency below 150 MHz. Set CEVCOD to 01 (Division factor of 4) for QE frequency above 150 MHz.
Section 4.4.1, “System Clock Domains”	Changed LCCR[CLKDIV] to LCRR[CLKDIV].
Table 4-21, Hard-Coded Reset Configuration Word High Field Values	Updated the number of bits in the third column.
System Boot	
Table 5-4, Config Address Field Description, CNT = 1 Table 5-7, Config Address Field Description, CNT = 1	For the DLY field, updated the clock from CCB to CSB.
Section 5.3.4, “SPI Controller Configuration” Figure 5-6, External Signal Connection	Changed signal name from $\overline{\text{SPI_CS}}$ to $\overline{\text{SPISEL_BOOT}}$.
System Configuration	
Table 6-28, SICR_2 Bit Settings for MPC8309	Changed the signal names HDLC2_RXCLK (CLK14) to HDLC2_RXCLK (CLK13) and HDLC2_TXCLK (CLK13) to HDLC2_TXCLK (CLK14). Modified the Pin Function 0 of HDLC2_B (at 20–21) to HDLC2_TXCLK(CLK14) and HDLC2_C (at 22–23) to HDLC2_RXCLK(CLK13).
Figure 6-15, System I/O Configuration Register 2 (SICR_2)	Replaced the pin name HDLC_C with HDLC2_C.
Table 6-35, GPR_1 Field Descriptions	Changed HDLC_RXD to HDLC1_RXD in the field description of PULLUP_CTRL_11 at bit 11.
Figure 6-14, System I/O Configuration Register 1 (SICR_1)	Changed reset from 0001_565F to 0000_000A.
Section 6.3.2.5, “System I/O Configuration Register 1 (SICR_1)”	Added the following note: “In case of eSDHC boot, the reset for SICR_1 changes from 0000_000A to 0028_000A.”

Table B-2. Changes from Revision 0 to Revision 1

Table 6-27, SICR_1 Bit Settings for MPC8309	Updated the Reset Value column as per updates in Figure 6-14.
Figure 6-15, System I/O Configuration Register 2 (SICR_2)	Changed reset from A005_0475 to: 9040_5500 (PCIARB=1) 9042_5500 (PCIARB=0)
Section 6.3.2.6, “System I/O Configuration Register 2 (SICR_2)”	Added the following note: “The reset value shown is for the case when PCIARB = 1. If PCIARB = 0, the reset for SICR_2 changes from 9040_5500 to 9042_5500.”
Table 6-28, SICR_2 Bit Settings for MPC8309	Updated the Reset Value column as per updates in Figure 6-15.
Table 6-20, System Configuration Register Memory Map	Updated the offset location 0x14C from Reserved to SPI_CS.
Section 6.3.2.13, “SPI Chip Select Register (SPI_CS)”	Added the section “SPI Chip Select Register (SPI_CS)”.
Table 6-1, Local Access Windows Target Interface	Updated target interface for window number 5–7. Also, added window number 8.
Table 6-20, System Configuration Register Memory Map	Removed System I/O delay configuration registers SIDCR0 and SIDCR1.
Section 6.3.2.9, “System I/O Delay Configuration Registers (SIDCR0–SIDCR1)”	Removed the section.
Table 6-27, SICR_1 Bit Settings for MPC8309	Added the following: <ul style="list-style-type: none"> • Signal QE_EXT_REQ_3 for bits 12-13, corresponding to GPIO[6] at pin function 3. • Signal QE_EXT_REQ_4 for bits 22-23, corresponding to USBDR_NXT at pin function 3.
Integrated Programmable Interrupt Controller (IPIC)	
Table 9-10, SIPNR_L/SIFCR_L/SIMSR_L Bit Assignments	Changed field descriptions of bits 1 and 2 to PIT and PCI respectively.
DDR Memory Controller	
Section 10.1, “Introduction”	Removed the following text from the first paragraph: “In addition, unbuffered and registered are supported. However, mixing different memory types or unbuffered and registered in the same system is not supported.”
Section 10.1, “Introduction”	Updated the first paragraph as follows: “The fully programmable DDR SDRAM controller supports most JEDEC standard × 8, and × 16, and × 32 DDR2 memories available. In addition, unbuffered and registered DRAM modules are supported. However, mixing different memory types or unbuffered and registered DRAM modules in the same system is not supported. Dynamic power management and auto-precharge modes simplify memory system design. A large set of special features support rapid system debug.”

Table B-2. Changes from Revision 0 to Revision 1

Figure 10-34, Example 64-Mbyte DDR SDRAM Configuration With ECC	Replaced all instances of MCK[0:2] with MCK[0:1] and $\overline{\text{MCK}}[0:2]$ with $\overline{\text{MCK}}[0:1]$.
Figure 10-38, DDR SDRAM Clock Distribution Example for $\times 8$ DDR SDRAMs	Removed MCK[2] and $\overline{\text{MCK}}[2]$ and updated the figure.
Enhanced Local Bus Controller	
Section 11.4.1.2, “External Address Latch Enable Signal (LALE)”	Updated the second sentence of the second paragraph as follows: “By default, LALE negates earlier by 1 platform clock period. For example, if the platform clock is operating at 133 Mhz, then 7.5 ns of address hold time is introduced.”
Section 11.4.1.2, “External Address Latch Enable Signal (LALE)”	Removed the following sentence: “Note that during each of the 32 assertions of LALE, LA[21:25] exactly mirror LAD[27:31], but during data phases, only LAD[0:7] is driven with valid data.”
Section 11.5.1.1, “Multiplexed Address/Data Bus for 26-Bit Addressing”	Removed the text “(with zero bits on LAD[16:31])” from the first sentence.
Figure 11-28, Example of 8-Bit GPCM Writing 32 Bytes to Address 0x5420 (LCRR[PBYP] = 0)	Updated the figure.
Table 11-21, LBCR Field Descriptions	Updated the description of LBCR[AHD] bit.
DMA Engine 2	
Section 14.2, “DMA External Signal Description”	Removed the section “DMA External Signal Description”
Section 14.5.3.1, “External Control”	Removed the section “External Control”.
Figure 14-10, DMA Mode Register (DMAMR _n) Table 14-10, DMAMR _n Field Descriptions	Changed DMAMR _n [EMSEN] bit field to “Reserved”. Also, removed references of EMSEN from bit fields DMAMR _n [DRCNT], DMAMR _n [SAHE], and DMAMR _n [DAHE].
Section 14.1, “DMA Features”	Removed the following bullet: “Optional external control signals (REQ/ACK/DONE) per channel”
Section 14.6.3, “Initialization Steps in Direct Mode with External Control	Removed the section “Initialization Steps in Direct Mode with External Control”.
Section 14.6.4, “Initialization Steps in Chaining Mode with External Control	Removed the section “Initialization Steps in Chaining Mode with External Control”.
FlexCAN	
Table 15-8, Module Configuration Register Description	In the description of bit “31”: Change From: “0 Disable the FlexCAN module 1 Enable the FlexCAN module” Change To: “0 Enable the FlexCAN module 1 Disable the FlexCAN module”

Table B-2. Changes from Revision 0 to Revision 1

Universal Serial Bus Interface	
Figure 16-34, USB General-Purpose Register (CONTROL) Table 16-36, CONTROL Field Descriptions	Added the bit "PHY_CLK_SEL" at position "21".
Serial Peripheral Interface	
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	Updated reset from "All zeros" to "All ones". And removed "Offset 0x36".
DUART	
Throughout	Replaced UART_CTS with UART1_CTS and UART_RTS with UART1_RTS.
General Purpose I/O (GPIO)	
Section 21.1, "Introduction"	Added the following note at the end of the section: "GPIO1[0:31] represents the signals GPIO[0] to GPIO[31], and GPIO2[0:31] represents signals GPIO[32] to GPIO[63] throughout this chapter."
PCI Bus Interface	
Table 23-3, PCI Interface Signals—Detailed Signal Descriptions	Updated the description of the $\overline{\text{PCI_INTA}}$ signal as follows: PCI interrupt A. It can be disabled from PCI Command Configuration Register. For more information, see Section 23.3.3.3, "PCI Command Configuration Register" and Table 14-6, ODR Field Descriptions .

