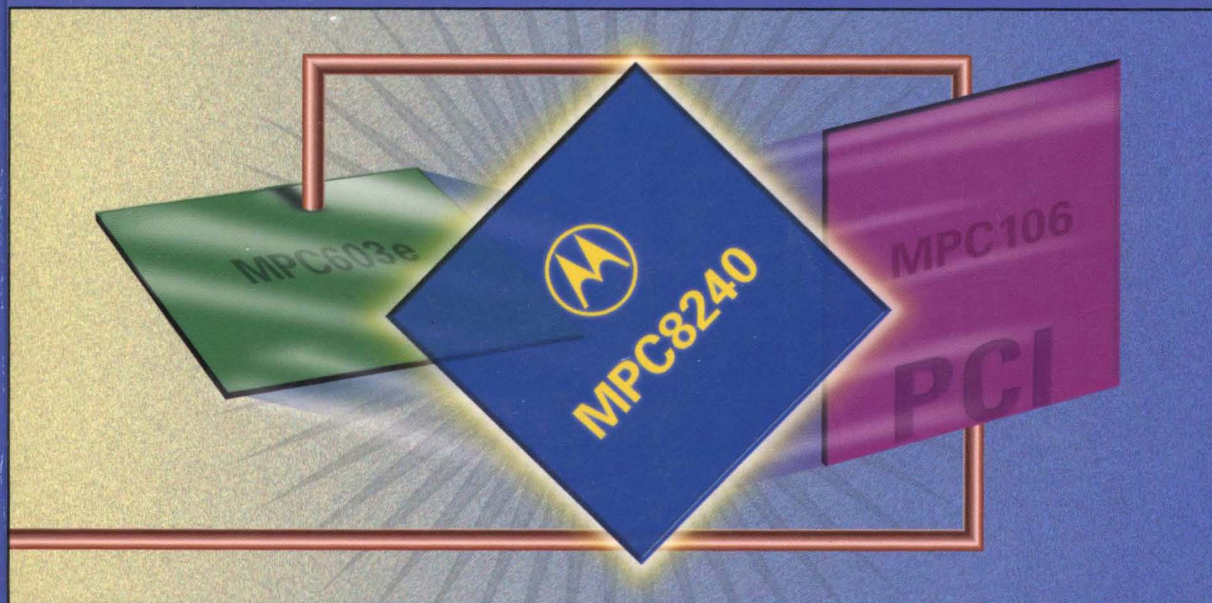


USER'S MANUAL



MPC8240 INTEGRATED PROCESSOR

MPC8240
Integrated Processor

user's manual

MPC8240UM/D
07/1999
Rev. 0

MPC8240 Integrated Processor User's Manual



PowerPC™




DigitalDNA and Mfax are trademarks of Motorola, Inc.

The PowerPC name, the PowerPC logotype, and PowerPC 603e are trademarks of International Business Machines Corporation used by Motorola under license from International Business Machines Corporation.

I²C is a registered trademark of Philips Semiconductors

This document contains information on a new product under development. Motorola reserves the right to change or discontinue this product without notice. Information in this document is provided solely to enable system and software implementers to use PowerPC microprocessors. There are no express or implied copyright licenses granted hereunder to design or fabricate PowerPC integrated circuits or integrated circuits based on the information in this document.

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola 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 can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola 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 Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola 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 Motorola was negligent regarding the design or manufacture of the part. Motorola and  are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

Motorola Literature Distribution Centers:

USA/EUROPE: Motorola Literature Distribution; P.O. Box 5405; Denver, Colorado 80217; Tel.: 1-800-441-2447 or 1-303-675-2140/

JAPAN: Nippon Motorola Ltd SPD, Strategic Planning Office 4-32-1, Nishi-Gotanda Shinagawa-ku, Tokyo 141, Japan Tel.: 81-3-5487-8488

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd.; 8B Tai Ping Industrial Park, 51 Ting Kok Road, Tai Po, N.T., Hong Kong; Tel.: 852-26629298

Mfax™: RMFAX0@email.sps.mot.com; TOUCHTONE 1-602-244-6609; US & Canada ONLY (800) 774-1848;

World Wide Web Address: <http://sps.motorola.com/mfax>

INTERNET: <http://motorola.com/sps>

Technical Information: Motorola Inc. SPS Customer Support Center 1-800-521-6274; electronic mail address: crc@wmkmail.sps.mot.com.

Document Comments: FAX (512) 895-2638, Attn: RISC Applications Engineering.

World Wide Web Addresses: <http://www.mot.com/PowerPC>

<http://www.mot.com/netcomm>

<http://www.mot.com/HPESD>

Overview	1
PowerPC Processor Core	2
Signal Descriptions and Clocking	3
Address Maps	4
Configuration Registers	5
MPC8240 Memory Interface	6
Central Control Unit	7
PCI Bus Interface	8
DMA Controller	9
Message Unit (I ₂ O)	10
I ² C Interface	11
Embedded Programmable Interrupt Controller (EPIC)	12
Error Handling and Exceptions	13
Power Management	14
Debug Features	15
PowerPC Instruction Set Listings	A
Bit and Byte Ordering	B
Glossary of Terms and Abbreviations	GLO
Index	IND

- 1** Overview
 - 2** PowerPC Processor Core
 - 3** Signal Descriptions and Clocking
 - 4** Address Maps
 - 5** Configuration Registers
 - 6** MPC8240 Memory Interface
 - 7** Central Control Unit
 - 8** PCI Bus Interface
 - 9** DMA Controller
 - 10** Message Unit (I₂O)
 - 11** I²C Interface
 - 12** Embedded Programmable Interrupt Controller (EPIC)
 - 13** Error Handling and Exceptions
 - 14** Power Management
 - 15** Debug Features
-
- A** PowerPC Instruction Set Listings
 - B** Bit and Byte Ordering
-
- GLO** Glossary of Terms and Abbreviations
 - IND** Index

CONTENTS

Paragraph Number	Title	Page Number
Chapter 1		
Overview		
1.1	MPC8240 Integrated Processor Overview.....	1-1
1.1.1	MPC8240 Integrated Processor Features.....	1-3
1.1.2	MPC8240 Integrated Processor Applications.....	1-5
1.2	Processor Core Overview	1-7
1.3	Peripheral Logic Bus.....	1-10
1.4	Peripheral Logic Overview	1-10
1.4.1	Peripheral Logic Features	1-11
1.4.2	Peripheral Logic Functional Units.....	1-12
1.4.3	Memory System Interface.....	1-13
1.4.4	Peripheral Component Interface (PCI)	1-14
1.4.4.1	PCI Bus Arbitration Unit	1-14
1.4.4.2	Address Maps and Translation	1-14
1.4.4.3	Byte Ordering	1-14
1.4.4.4	PCI Agent Capability.....	1-15
1.4.5	DMA Controller.....	1-15
1.4.6	Message Unit (MU)	1-15
1.4.6.1	Doorbell Registers	1-15
1.4.6.2	Inbound and Outbound Message Registers	1-15
1.4.6.3	Intelligent Input/Output Controller (I ₂ O)	1-16
1.4.7	Inter-Integrated Circuit (I ² C) Controller	1-16
1.4.8	Embedded Programmable Interrupt Controller (EPIC).....	1-16
1.4.9	Integrated PCI Bus and SDRAM Clock Generation	1-17
1.4.10	Bus Ratios	1-17
1.5	Power Management	1-17
1.5.1	Programmable Processor Power Management Modes	1-18
1.5.2	Peripheral Logic Power Management Modes.....	1-18
1.6	Debug Features	1-19
1.6.1	Memory Attribute and PCI Attribute Signals	1-19
1.6.2	Memory Debug Address.....	1-19
1.6.3	Memory Interface Valid (\overline{MIV})	1-20
1.6.4	Error Injection/Capture on Data Path	1-20
1.6.5	IEEE 1149.1 (JTAG)/Test Interface	1-20

CONTENTS

Paragraph Number	Title	Page Number
Chapter 2		
PowerPC Processor Core		
2.1	Overview	2-1
2.2	PowerPC Processor Core Features	2-3
2.2.1	Instruction Unit.....	2-5
2.2.2	Instruction Queue and Dispatch Unit	2-5
2.2.3	Branch Processing Unit (BPU).....	2-6
2.2.4	Independent Execution Units	2-6
2.2.4.1	Integer Unit (IU).....	2-6
2.2.4.2	Floating-Point Unit (FPU).....	2-7
2.2.4.3	Load/Store Unit (LSU).....	2-7
2.2.4.4	System Register Unit (SRU)	2-7
2.2.5	Completion Unit.....	2-8
2.2.6	Memory Subsystem Support	2-8
2.2.6.1	Memory Management Units (MMUs)	2-8
2.2.6.2	Cache Units	2-8
2.2.6.3	Peripheral Logic Bus Interface.....	2-9
2.2.6.3.1	Peripheral Logic Bus Protocol	2-9
2.2.6.3.2	Peripheral Logic Bus Data Transfers	2-9
2.2.6.3.3	Peripheral Logic Bus Frequency	2-10
2.3	Programming Model.....	2-10
2.3.1	Register Set.....	2-10
2.3.1.1	PowerPC Register Set	2-11
2.3.1.2	MPC8240-Specific Registers	2-13
2.3.1.2.1	Hardware Implementation-Dependent Register 0 (HID0)	2-13
2.3.1.2.2	Hardware Implementation-Dependent Register 1 (HID1)	2-17
2.3.1.2.3	Hardware Implementation-Dependent Register 2 (HID2)	2-17
2.3.1.2.4	Processor Version Register (PVR)	2-18
2.3.2	PowerPC Instruction Set and Addressing Modes.....	2-18
2.3.2.1	Calculating Effective Addresses	2-18
2.3.2.2	PowerPC Instruction Set	2-18
2.3.2.3	MPC8240 Implementation-Specific Instruction Set	2-20
2.4	Cache Implementation.....	2-20
2.4.1	PowerPC Cache Model.....	2-21
2.4.2	MPC8240 Implementation-Specific Cache Implementation.....	2-21
2.4.2.1	Data Cache	2-21
2.4.2.2	Instruction Cache.....	2-23
2.4.2.3	Cache Locking.....	2-23
2.4.2.3.1	Entire Cache Locking.....	2-23
2.4.2.3.2	Way Locking	2-23
2.4.3	Cache Coherency.....	2-24
2.4.3.1	CCU Responses to Processor Transactions.....	2-24

CONTENTS

Paragraph Number	Title	Page Number
2.4.3.2	Processor Responses to PCI-to-Memory Transactions	2-25
2.5	Exception Model	2-26
2.5.1	PowerPC Exception Model	2-26
2.5.2	MPC8240 Implementation-Specific Exception Model	2-28
2.5.3	Exception Priorities	2-31
2.6	Memory Management	2-31
2.6.1	PowerPC MMU Model	2-31
2.6.2	MPC8240 Implementation-Specific MMU Features	2-32
2.7	Instruction Timing	2-33
2.8	Differences between the MPC8240 Core and the PowerPC 603e Microprocessor	2-34

Chapter 3 Signal Descriptions and Clocking

3.1	Signal Overview	3-1
3.1.1	Signal Cross Reference	3-4
3.1.2	Output Signal States at Reset	3-7
3.2	Detailed Signal Descriptions	3-7
3.2.1	PCI Interface Signals	3-7
3.2.1.1	PCI Address/Data Bus (AD[31-0])	3-8
3.2.1.1.1	Address/Data (AD[31-0])—Output	3-8
3.2.1.1.2	Address/Data (AD[31-0])—Input	3-8
3.2.1.2	Command/Byte Enable ($\overline{C/BE}$ [3-0])	3-8
3.2.1.2.1	Command/Byte Enable ($\overline{C/BE}$ [3-0])—Output	3-8
3.2.1.2.2	Command/Byte Enable ($\overline{C/BE}$ [3-0])—Input	3-9
3.2.1.3	Device Select (\overline{DEVSEL})	3-9
3.2.1.3.1	Device Select (\overline{DEVSEL})—Output	3-9
3.2.1.3.2	Device Select (\overline{DEVSEL})—Input	3-10
3.2.1.4	Frame (\overline{FRAME})	3-10
3.2.1.4.1	Frame (\overline{FRAME})—Output	3-10
3.2.1.4.2	Frame (\overline{FRAME})—Input	3-10
3.2.1.5	PCI Bus Grant (\overline{GNT} [4-0])—Output	3-10
3.2.1.5.1	PCI Bus Grant (\overline{GNT} [4-0])—Internal Arbiter Enabled	3-10
3.2.1.5.2	PCI Bus Grant (\overline{GNT} [4-0])—Internal Arbiter Disabled	3-11
3.2.1.6	PCI Bus Request (\overline{REQ} [4-0])—Input	3-11
3.2.1.6.1	PCI Bus Request (\overline{REQ} [4-0])—Internal Arbiter Enabled	3-11
3.2.1.6.2	PCI Bus Request (\overline{REQ} [4-0])—Internal Arbiter Disabled	3-11
3.2.1.7	Initiator Ready (\overline{IRDY})	3-12
3.2.1.7.1	Initiator Ready (\overline{IRDY})—Output	3-12
3.2.1.7.2	Initiator Ready (\overline{IRDY})—Input	3-12
3.2.1.8	Lock (\overline{LOCK})—Input	3-12

CONTENTS

Paragraph Number	Title	Page Number
3.2.1.9	Parity (PAR)	3-13
3.2.1.9.1	Parity (PAR)—Output	3-13
3.2.1.9.2	Parity (PAR)—Input	3-13
3.2.1.10	Parity Error ($\overline{\text{PERR}}$)	3-13
3.2.1.10.1	Parity Error ($\overline{\text{PERR}}$)—Output	3-13
3.2.1.10.2	Parity Error ($\overline{\text{PERR}}$)—Input	3-13
3.2.1.11	System Error ($\overline{\text{SERR}}$)	3-13
3.2.1.11.1	System Error ($\overline{\text{SERR}}$)—Output	3-14
3.2.1.11.2	System Error ($\overline{\text{SERR}}$)—Input	3-14
3.2.1.12	Stop ($\overline{\text{STOP}}$)	3-14
3.2.1.12.1	Stop ($\overline{\text{STOP}}$)—Output	3-14
3.2.1.12.2	Stop ($\overline{\text{STOP}}$)—Input	3-14
3.2.1.13	Target Ready ($\overline{\text{TRDY}}$)	3-14
3.2.1.13.1	Target Ready ($\overline{\text{TRDY}}$)—Output	3-14
3.2.1.13.2	Target Ready ($\overline{\text{TRDY}}$)—Input	3-15
3.2.1.14	Interrupt Request ($\overline{\text{INTA}}$)—Output	3-15
3.2.1.15	ID Select ($\overline{\text{IDSEL}}$)—Input	3-15
3.2.2	Memory Interface Signals	3-16
3.2.2.1	Row Address Strobe ($\overline{\text{RAS}}[0-7]$)—Output	3-16
3.2.2.2	Column Address Strobe ($\overline{\text{CAS}}[0-7]$)—Output	3-16
3.2.2.3	SDRAM Command Select ($\overline{\text{CS}}[0-7]$)—Output	3-16
3.2.2.4	SDRAM Data Qualifier ($\text{DQM}[0-7]$)—Output	3-17
3.2.2.5	Write Enable ($\overline{\text{WE}}$)—Output	3-17
3.2.2.6	SDRAM Address ($\text{SDMA}[11-0]$)—Output	3-17
3.2.2.7	SDRAM Address 12 ($\text{SDMA}12$)—Output	3-18
3.2.2.8	SDRAM Internal Bank Select 1 ($\text{SDBA}1$)—Output	3-18
3.2.2.9	SDRAM Internal Bank Select 0 ($\text{SDBA}0$)—Output	3-18
3.2.2.10	Data Bus ($\text{DH}[0-31]$, $\text{DL}[0-31]$)	3-18
3.2.2.10.1	Data Bus ($\text{DH}[0-31]$, $\text{DL}[0-31]$)—Output	3-19
3.2.2.10.2	Data Bus ($\text{DH}[0-31]$, $\text{DL}[0-31]$)—Input	3-19
3.2.2.11	Data Parity/ECC ($\text{PAR}[0-7]$)	3-19
3.2.2.11.1	Data Parity ($\text{PAR}[0-7]$)—Output	3-19
3.2.2.11.2	Data Parity ($\text{PAR}[0-7]$)—Input	3-20
3.2.2.12	ROM Address 19–12 ($\text{AR}[19-12]$)—Output	3-20
3.2.2.13	SDRAM Clock Enable (CKE)—Output	3-20
3.2.2.14	SDRAM Row Address Strobe ($\overline{\text{SDRAS}}$)—Output	3-21
3.2.2.15	SDRAM Column Address Strobe ($\overline{\text{SDCAS}}$)—Output	3-21
3.2.2.16	ROM Bank 0 Select ($\overline{\text{RCS}}0$)—Output	3-21
3.2.2.17	ROM Bank 1 Select ($\overline{\text{RCS}}1$)—Output	3-21
3.2.2.18	Flash Output Enable ($\overline{\text{FOE}}$)—Output	3-22
3.2.2.19	Address Strobe ($\overline{\text{AS}}$)—Output	3-22
3.2.3	EPIC Control Signals	3-22
3.2.3.1	Discrete Interrupt 0–4 ($\text{IRQ}[0-4]$)—Input	3-22

CONTENTS

Paragraph Number	Title	Page Number
3.2.3.2	Serial Interrupt Mode Signals	3-23
3.2.3.2.1	Serial Interrupt Stream (S_INT)—Input	3-23
3.2.3.2.2	Serial Interrupt Clock (S_CLK)—Output	3-23
3.2.3.2.3	Serial Interrupt Reset (S_RST)—Output	3-23
3.2.3.2.4	Serial Interrupt Frame (S_FRAME)—Output	3-23
3.2.3.3	Local Interrupt (L_INT)—Output	3-23
3.2.4	I ² C Interface Control Signals	3-23
3.2.4.1	Serial Data (SDA)	3-24
3.2.4.1.1	Serial Data (SDA)—Output	3-24
3.2.4.1.2	Serial Data (SDA)—Input	3-24
3.2.4.2	Serial Clock (SCL)	3-24
3.2.4.2.1	Serial Clock (SCL)—Output	3-24
3.2.4.2.2	Serial Clock (SCL)—Input	3-24
3.2.5	System Control and Power Management Signals	3-24
3.2.5.1	Hard Reset	3-24
3.2.5.1.1	Hard Reset (Processor) (HRST_CPU)—Input	3-25
3.2.5.1.2	Hard Reset (Peripheral Logic) (HRST_CTRL)—Input	3-25
3.2.5.2	Soft Reset (SRESET)—Input	3-25
3.2.5.3	Machine Check (MCP)—Output	3-25
3.2.5.4	Nonmaskable Interrupt (NMI)—Input	3-26
3.2.5.5	System Management Interrupt (SMI)—Input	3-26
3.2.5.6	Checkstop In (CHKSTOP_IN)—Input	3-27
3.2.5.7	Time Base Enable (TBEN)—Input	3-27
3.2.5.8	Quiesce Acknowledge (QACK)—Output	3-27
3.2.5.9	Debug Signals	3-27
3.2.5.9.1	Memory Address Attributes (MAA[0–2])—Output	3-27
3.2.5.9.2	PCI Address Attributes (PMAA[0–2])—Output	3-28
3.2.5.9.3	Debug Address (DA[0–15])—Output	3-28
3.2.5.9.4	Memory Interface Valid (MIV)—Input	3-28
3.2.6	Test and Configuration Signals	3-29
3.2.6.1	PLL Configuration (PLL_CFG[0–4])—Input	3-29
3.2.6.2	JTAG Test Clock (TCK)—Input	3-29
3.2.6.3	JTAG Test Data Input (TDI)—Input	3-29
3.2.6.4	JTAG Test Data Output (TDO)—Output	3-30
3.2.6.5	JTAG Test Mode Select (TMS)—Input	3-30
3.2.6.6	JTAG Test Reset (TRST)—Input	3-30
3.2.7	Clock Signals	3-30
3.2.7.1	System Clock Input (OSC_IN)—Input	3-30
3.2.7.2	PCI Clock (PCI_CLK[0–4])—Output	3-30
3.2.7.3	PCI Clock Synchronize Out (PCI_SYNC_OUT)—Output	3-31
3.2.7.4	PCI Feedback Clock (PCI_SYNC_IN)—Input	3-31
3.2.7.5	SDRAM Clock Outputs (SDRAM_CLK[0–3])—Output	3-31
3.2.7.6	SDRAM Clock Synchronize Out (SDRAM_SYNC_OUT)—Output	3-31

CONTENTS

Paragraph Number	Title	Page Number
3.2.7.7	SDRAM Feedback Clock (SDRAM_SYNC_IN)—Input.....	3-31
3.2.7.8	Debug Clock (CKO)—Output.....	3-31
3.3	Clocking	3-32
3.3.1	Clocking Method.....	3-32
3.3.2	DLL Operation and Locking	3-33
3.3.3	Clock Synchronization	3-34
3.3.4	Clocking System Solution Examples	3-34
3.4	Configuration Pins Sampled at Reset	3-35

Chapter 4 Address Maps

4.1	Address Map A	4-1
4.2	Address Map B	4-5
4.3	Address Translation.....	4-10
4.3.1	Inbound PCI Address Translation	4-10
4.3.2	Outbound PCI Address Translation.....	4-11
4.3.3	Address Translation Registers.....	4-12
4.3.3.1	Local Memory Base Address Register (LMBAR).....	4-13
4.3.3.2	Inbound Translation Window Register (ITWR)	4-14
4.3.3.3	Outbound Memory Base Address Register (OMBAR).....	4-15
4.3.3.4	Outbound Translation Window Register (OTWR)	4-15
4.4	Embedded Utilities Memory Block (EUMB).....	4-16
4.4.1	Processor Core Control and Status Registers	4-17
4.4.2	Peripheral Control and Status Registers	4-18

Chapter 5 Configuration Registers

5.1	Configuration Register Access	5-1
5.1.1	Processor Access to Configuration Registers (Map A)	5-1
5.1.1.1	Indirect Access Method.....	5-1
5.1.1.2	Direct Access Method	5-2
5.1.2	Processor Access to Configuration Registers (Map B)	5-2
5.1.3	PCI Access to Configuration Registers	5-2
5.1.4	Configuration Register Access in Little-Endian Mode	5-2
5.1.4.1	Configuration Register Access in Big-Endian Mode	5-3
5.1.5	Configuration Register Summary.....	5-4
5.1.5.1	Processor-Accessible Configuration Registers	5-4
5.1.5.2	PCI-Accessible Configuration Registers	5-6
5.1.6	Configuration Register Order	5-8
5.2	PCI Interface Configuration Registers	5-8

CONTENTS

Paragraph Number	Title	Page Number
5.2.1	PCI Command Register.....	5-10
5.2.2	PCI Status Register	5-11
5.2.3	Programming Interface.....	5-12
5.2.4	PCI Base Class Code.....	5-13
5.2.5	PCI Cache Line Size	5-13
5.2.6	Latency Timer	5-13
5.2.7	PCI Base Address Registers.....	5-13
5.2.8	PCI Interrupt Line	5-14
5.2.9	PCI Arbiter Control Register	5-15
5.3	Peripheral Logic Power Management Configuration Registers (PMCRs)	5-15
5.3.1	Power Management Configuration Register 1 (PMCR1)	5-15
5.3.2	Power Management Configuration Register 2 (PMCR2)	5-17
5.4	Output Driver Configuration Registers	5-18
5.5	Embedded Utilities Memory Block Base Address Register	5-21
5.6	Processor Interface Configuration Registers.....	5-22
5.7	Error Handling Registers.....	5-26
5.7.1	ECC Single-Bit Error Registers	5-26
5.7.2	Error Enabling Registers	5-27
5.7.3	Error Detection Registers	5-29
5.7.4	Error Status Registers.....	5-31
5.8	Address Map B Options Register.....	5-33
5.9	Memory Interface Configuration Registers.....	5-34
5.9.1	Memory Boundary Registers	5-35
5.9.2	Memory Bank Enable Register	5-39
5.9.3	Memory Page Mode Register.....	5-40
5.9.4	Memory Control Configuration Registers	5-41

Chapter 6 MPC8240 Memory Interface

6.1	Memory Interface Signal Summary	6-4
6.2	Memory Interface Configuration at Reset.....	6-6
6.3	FPM or EDO DRAM Interface Operation	6-6
6.3.1	Supported FPM or EDO DRAM Organizations	6-9
6.3.2	FPM or EDO DRAM Address Multiplexing	6-11
6.3.2.1	Row Bit Multiplexing During (RAS) The Row Phase	6-11
6.3.2.2	Column Bit Multiplexing During (CAS) the Column Phase	6-12
6.3.2.3	Graphical View of the Row and Column Bit Multiplexing	6-13
6.3.3	FPM or EDO Memory Data Interface.....	6-14
6.3.4	FPM or EDO DRAM Initialization.....	6-16
6.3.5	FPM or EDO DRAM Interface Timing	6-17
6.3.6	DMA Burst Wrap.....	6-22

CONTENTS

Paragraph Number	Title	Page Number
6.3.7	FPM or EDO DRAM Page Mode Retention.....	6-23
6.3.8	FPM or EDO DRAM Parity and RMW Parity.....	6-23
6.3.8.1	RMW Parity Latency Considerations.....	6-24
6.3.9	FPM or EDO ECC.....	6-24
6.3.9.1	FPM or EDO DRAM Interface Timing with ECC.....	6-26
6.3.10	FPM or EDO DRAM Refresh.....	6-28
6.3.10.1	FPM or EDO Refresh Timing.....	6-28
6.3.11	FPM or EDO DRAM Power Saving Modes.....	6-29
6.3.11.1	Self-Refresh in Sleep Mode.....	6-30
6.4	SDRAM Interface Operation.....	6-31
6.4.1	Supported SDRAM Organizations.....	6-35
6.4.2	SDRAM Address Multiplexing.....	6-36
6.4.3	SDRAM Memory Data Interface.....	6-39
6.4.4	SDRAM ECC.....	6-41
6.4.5	SDRAM Burst and Single-Beat Transactions.....	6-43
6.4.6	SDRAM Page Mode Retention.....	6-44
6.4.6.1	SDRAM Paging in Sleep Mode.....	6-46
6.4.7	SDRAM Power on Initialization.....	6-46
6.4.8	MPC8240 Interface Functionality for JEDEC SDRAMs.....	6-47
6.4.9	SDRAM Interface Timing.....	6-49
6.4.9.1	SDRAM Mode-Set Command Timing.....	6-53
6.4.10	SDRAM Parity and RMW Parity.....	6-53
6.4.10.1	RMW Parity Latency Considerations.....	6-54
6.4.11	SDRAM and In-Line ECC or Parity.....	6-55
6.4.12	SDRAM Registered DIMM Mode.....	6-55
6.4.13	SDRAM Refresh.....	6-56
6.4.13.1	SDRAM Refresh Timing.....	6-57
6.4.13.2	SDRAM Refresh and Power Saving Modes.....	6-58
6.5	ROM/Flash Interface Operation.....	6-61
6.5.1	ROM/Flash Address Multiplexing.....	6-65
6.5.2	64 or 32-Bit ROM/Flash Interface Timing.....	6-66
6.5.3	8-Bit ROM/Flash Interface Timing.....	6-69
6.5.4	ROM/Flash Interface Write Operations.....	6-70
6.5.5	ROM/Flash Interface Write Timing.....	6-71
6.5.6	Port X Interface.....	6-72

Chapter 7 Central Control Unit

7.1	Internal Buffers.....	7-1
7.1.1	Processor Core/System Memory Buffers.....	7-3
7.1.2	Processor/PCI Buffers.....	7-4

CONTENTS

Paragraph Number	Title	Page Number
7.1.2.1	Processor-to-PCI-Read Buffer (PRPRB)	7-4
7.1.2.2	Processor-to-PCI-Write Buffers (PRPWBs)	7-5
7.1.3	PCI/System Memory Buffers	7-6
7.1.3.1	PCI to System Memory Read Buffering	7-7
7.1.3.2	PCI-to-System-Memory-Read Buffers (PCMRBs)	7-7
7.1.3.3	Speculative PCI Reads from System Memory	7-8
7.1.3.4	PCI-to-System-Memory-Write Buffers (PCMWBs)	7-8
7.2	Internal Arbitration.....	7-9

Chapter 8 PCI Bus Interface

8.1	PCI Interface Overview.....	8-1
8.1.1	The MPC8240 as a PCI Initiator	8-2
8.1.2	The MPC8240 as a PCI Target	8-3
8.1.3	PCI Signal Output Hold Timing	8-3
8.2	PCI Bus Arbitration.....	8-4
8.2.1	PCI Bus Arbiter Operation.....	8-4
8.2.2	PCI Bus Parking	8-6
8.2.3	Broken Master Lock-Out	8-6
8.2.4	Bus Lock Mode	8-6
8.2.5	Power-Saving Modes and the PCI Arbiter	8-6
8.3	PCI Bus Protocol.....	8-7
8.3.1	Basic Transfer Control	8-7
8.3.2	PCI Bus Commands	8-8
8.3.3	Addressing.....	8-9
8.3.3.1	Memory Space Addressing	8-10
8.3.3.2	I/O Space Addressing.....	8-10
8.3.3.3	Configuration Space Addressing	8-10
8.3.4	Device Selection.....	8-10
8.3.5	Byte Alignment	8-11
8.3.6	Bus Driving and Turnaround	8-11
8.4	PCI Bus Transactions	8-12
8.4.1	Read Transactions	8-12
8.4.2	Write Transactions	8-13
8.4.3	Transaction Termination	8-14
8.4.3.1	Master-Initiated Termination	8-14
8.4.3.2	Target-Initiated Termination	8-15
8.4.4	Fast Back-to-Back Transactions	8-17
8.4.5	Configuration Cycles	8-18
8.4.5.1	The PCI Configuration Space Header.....	8-18
8.4.5.2	Accessing the PCI Configuration Space	8-19

CONTENTS

Paragraph Number	Title	Page Number
8.4.6	Other Bus Transactions	8-24
8.4.6.1	Interrupt Acknowledge Transactions	8-24
8.4.6.2	Special-Cycle Transactions	8-25
8.5	Exclusive Access	8-26
8.5.1	Starting an Exclusive Access.....	8-26
8.5.2	Continuing an Exclusive Access	8-27
8.5.3	Completing an Exclusive Access	8-27
8.5.4	Attempting to Access a Locked Target	8-27
8.5.5	Exclusive Access and the MPC8240	8-27
8.6	PCI Error Functions.....	8-28
8.6.1	PCI Parity	8-28
8.6.2	Error Reporting.....	8-29
8.7	PCI Host and Agent Modes	8-29
8.7.1	PCI Initialization Options.....	8-30
8.7.2	Accessing the MPC8240 Configuration Space in Agent Mode	8-31
8.7.3	PCI Configuration Cycle Retry Capability in Agent Mode	8-31
8.7.4	PCI Address Translation Support.....	8-31
8.7.4.1	Inbound PCI Address Translation	8-31
8.7.4.2	Outbound PCI Address Translation	8-32
8.7.4.3	Initialization Code Translation in Agent Mode.....	8-32

Chapter 9 DMA Controller

9.1	DMA Overview	9-1
9.2	DMA Register Summary	9-2
9.3	DMA Operation.....	9-3
9.3.1	DMA Direct Mode	9-4
9.3.2	DMA Chaining Mode.....	9-4
9.3.2.1	Basic Chaining Mode Initialization.....	9-4
9.3.2.2	Periodic DMA Feature	9-5
9.3.3	DMA Coherency	9-5
9.4	DMA Transfer Types.....	9-6
9.4.1	PCI to PCI.....	9-6
9.4.2	PCI to Local Memory	9-6
9.4.3	Local Memory to PCI.....	9-6
9.4.4	Local Memory to Local Memory	9-6
9.4.5	Address Map Interactions.....	9-7
9.4.5.1	Host Mode Interactions	9-7
9.4.5.1.1	PCI Master Abort when PCI Bus Specified for Lower 2-Gbyte Space	9-7
9.4.5.1.2	Address Alias to Lower 2-Gbyte Space	9-7
9.4.5.1.3	Attempted Writes to Local ROM/Port X Space	9-7

CONTENTS

Paragraph Number	Title	Page Number
9.4.5.1.4	Attempted Accesses to ROM on the PCI Bus—Host Mode.....	9-7
9.4.5.1.5	Attempted Accesses to ROM on the Memory Bus.....	9-8
9.4.5.2	Agent Mode Interactions.....	9-8
9.4.5.2.1	Agent Mode DMA Transfers for PCI.....	9-8
9.4.5.2.2	Attempted Accesses to Local ROM when ROM is on PCI.....	9-8
9.4.5.2.3	Attempted Access to ROM on the PCI Bus - Agent Mode.....	9-8
9.5	DMA Descriptors.....	9-8
9.5.1	Descriptors in Big-Endian Mode.....	9-11
9.5.2	Descriptors in Little-Endian Mode.....	9-11
9.6	DMA Register Descriptions.....	9-12
9.6.1	DMA Mode Registers (DMRs).....	9-12
9.6.2	DMA Status Registers (DSRs).....	9-14
9.6.3	Current Descriptor Address Registers (CDARs).....	9-15
9.6.4	Source Address Registers (SARs).....	9-16
9.6.5	Destination Address Registers (DARs).....	9-17
9.6.6	Byte Count Registers (BCRs).....	9-18
9.6.7	Next Descriptor Address Registers (NDARs).....	9-18

Chapter 10 Message Unit (with I₂O)

10.1	Message Unit (MU) Overview.....	10-1
10.2	Message and Doorbell Register Programming Model.....	10-1
10.2.1	Message and Doorbell Register Summary.....	10-1
10.2.2	Message Register Descriptions.....	10-2
10.2.3	Doorbell Register Descriptions.....	10-3
10.3	I ₂ O Interface.....	10-4
10.3.1	PCI Configuration Identification.....	10-4
10.3.2	I ₂ O Register Summary.....	10-5
10.3.3	FIFO Descriptions.....	10-5
10.3.3.1	Inbound FIFOs.....	10-6
10.3.3.1.1	Inbound Free_List FIFO.....	10-7
10.3.3.1.2	Inbound Post_List FIFO.....	10-7
10.3.3.2	Outbound FIFOs.....	10-7
10.3.3.2.1	Outbound Free_List FIFO.....	10-7
10.3.3.2.2	Outbound Post_List FIFO.....	10-8
10.3.4	I ₂ O Register Descriptions.....	10-8
10.3.4.1	PCI-Accessible I ₂ O Registers.....	10-8
10.3.4.1.1	Outbound Message Interrupt Status Register (OMISR).....	10-8
10.3.4.1.2	Outbound Message Interrupt Mask Register (OMIMR).....	10-9
10.3.4.1.3	Inbound FIFO Queue Port Register (IFQPR).....	10-10
10.3.4.1.4	Outbound FIFO Queue Port Register (OFQPR).....	10-11

CONTENTS

Paragraph Number	Title	Page Number
10.3.4.2	Processor-Accessible I ₂ O Registers	10-11
10.3.4.2.1	Inbound Message Interrupt Status Register (IMISR)	10-11
10.3.4.2.2	Inbound Message Interrupt Mask Register (IMIMR)	10-13
10.3.4.2.3	Inbound Free_FIFO Head Pointer Register (IFHPR)	10-14
10.3.4.2.4	Inbound Free_FIFO Tail Pointer Register (IFTPR)	10-15
10.3.4.2.5	Inbound Post_FIFO Head Pointer Register (IPHPR)	10-15
10.3.4.2.6	Inbound Post_FIFO Tail Pointer Register (IPTPR)	10-16
10.3.4.2.7	Outbound Free_FIFO Head Pointer Register (OFHPR)	10-17
10.3.4.2.8	Outbound Free_FIFO Tail Pointer Register (OFTPR)	10-17
10.3.4.2.9	Outbound Post_FIFO Head Pointer Register (OPHPR)	10-18
10.3.4.2.10	Outbound Post_FIFO Tail Pointer Register (OPTPR)	10-18
10.3.4.2.11	Messaging Unit Control Register (MUCR)	10-19
10.3.4.2.12	Queue Base Address Register (QBAR)	10-20

Chapter 11 I²C Interface

11.1	I ² C Interface Overview	11-1
11.1.1	I ² C Unit Features	11-2
11.1.2	I ² C Interface Signal Summary	11-2
11.1.3	I ² C Block Diagram	11-3
11.2	I ² C Protocol	11-3
11.2.1	START Condition	11-4
11.2.2	Slave Address Transmission	11-4
11.2.3	Data Transfer	11-5
11.2.4	Repeated START Condition	11-5
11.2.5	STOP Condition	11-5
11.2.6	Arbitration Procedure	11-5
11.2.7	Clock Synchronization	11-6
11.2.8	Handshaking	11-6
11.2.9	Clock Stretching	11-7
11.3	Programming Model	11-7
11.3.1	I ² C Address Register (I2CADR)	11-7
11.3.2	I ² C Frequency Divider Register (I2CFDR)	11-8
11.3.3	I ² C Control Register (I2CCR)	11-10
11.3.4	I ² C Status Register (I2CSR)	11-12
11.3.5	I ² C Data Register (I2CDR)	11-13
11.4	Programming Guidelines	11-14
11.4.1	Initialization Sequence	11-14
11.4.2	Generation of START	11-15
11.4.3	Post-Transfer Software Response	11-15
11.4.4	Generation of STOP	11-16

CONTENTS

Paragraph Number	Title	Page Number
11.4.5	Generation of Repeated START	11-16
11.4.6	Slave Mode Interrupt Service Routine	11-16
11.4.6.1	Slave Transmitter and Received Acknowledge	11-16
11.4.6.2	Loss of Arbitration and Forcing of Slave Mode	11-17
11.4.7	Interrupt Service Routine Flowchart	11-17

Chapter 12 Embedded Programmable Interrupt Controller (EPIC)

12.1	EPIC Unit Overview	12-1
12.1.1	EPIC Features Summary	12-2
12.1.2	EPIC Interface Signal Description	12-2
12.1.3	EPIC Block Diagram	12-3
12.2	EPIC Register Summary	12-3
12.3	EPIC Unit Interrupt Protocol	12-7
12.3.1	Interrupt Source Priority	12-7
12.3.2	Processor Current Task Priority	12-7
12.3.3	Interrupt Acknowledge	12-7
12.3.4	Nesting of Interrupts	12-7
12.3.5	Spurious Vector Generation	12-8
12.3.6	Internal Block Diagram Description	12-8
12.3.6.1	Interrupt Pending Register (IPR)—Non-programmable	12-9
12.3.6.2	Interrupt Selector (IS)	12-9
12.3.6.3	Interrupt Request Register (IRR)	12-9
12.3.6.4	In-Service Register (ISR)	12-9
12.4	EPIC Pass-Through Mode	12-10
12.5	EPIC Direct Interrupt Mode	12-10
12.6	EPIC Serial Interrupt Interface	12-10
12.6.1	Sampling of Serial Interrupts	12-11
12.6.2	Edge/Level Sensitivity of Serial Interrupts	12-11
12.6.3	Serial Interrupt Timing Protocol	12-12
12.7	EPIC Timers	12-12
12.8	Programming Guidelines	12-13
12.9	Register Definitions	12-15
12.9.1	Feature Reporting Register (FPR)	12-15
12.9.2	Global Configuration Register (GCR)	12-15
12.9.3	EPIC Interrupt Configuration Register (EICR)	12-16
12.9.4	EPIC Vendor Identification Register (EVI)	12-17
12.9.5	Processor Initialization Register (PI)	12-18
12.9.6	Spurious Vector Register (SVR)	12-18
12.9.7	Global Timer Registers	12-19
12.9.7.1	Timer Frequency Reporting Register (TFRR)	12-19

CONTENTS

Paragraph Number	Title	Page Number
12.9.7.2	Global Timer Current Count Registers (GTCCRs)	12-20
12.9.7.3	GlobalTimer Base Count Registers (GTBCRs)	12-20
12.9.7.4	Global Timer Vector/Priority Registers (GTVPRs)	12-21
12.9.7.5	Global Timer Destination Registers (GTDRs)	12-22
12.9.8	Direct, Serial, and Internal Interrupt Registers	12-22
12.9.8.1	Direct & Serial Interrupt Vector/Priority Regs (IVPRs, SVPRs)	12-22
12.9.8.2	Direct & Serial Interrupt Destination Registers (IDRs, SDRs)	12-24
12.9.8.3	Internal (I ² C, DMA, I ₂ O) Interrupt Vector/Priority Regs (IIVPRs)	12-24
12.9.8.4	Internal (I ² C, DMA or I ₂ O) Interrupt Destination Regs (IIDRs)	12-24
12.9.9	Processor-Related Registers	12-24
12.9.9.1	Processor Current Task Priority Register (PCTPR)	12-25
12.9.9.2	Processor Interrupt Acknowledge Register (IACK)	12-25
12.9.10	Processor End-of-Interrupt Register (EOI)	12-26

Chapter 13 Error Handling and Exceptions

13.1	Overview	13-1
13.1.1	Error Handling Block Diagram	13-2
13.1.2	Priority of Externally-Generated Errors and Exceptions	13-2
13.2	Exceptions and Error Signals	13-3
13.2.1	System Reset	13-3
13.2.2	Processor Core Error Signal (mcp)	13-3
13.2.3	PCI Bus Error Signals	13-4
13.2.3.1	System Error (SERR)	13-4
13.2.3.2	Parity Error (PERR)	13-4
13.2.3.3	Nonmaskable Interrupt (NMI)	13-5
13.3	Error Reporting	13-5
13.3.1	Processor Interface	13-6
13.3.1.1	Processor Transaction Error	13-6
13.3.1.2	Flash Write Error	13-6
13.3.2	Memory Interface	13-6
13.3.2.1	System Memory Read Data Parity Error	13-7
13.3.2.2	System Memory ECC Error	13-7
13.3.2.3	System Memory Select Error	13-7
13.3.2.4	System Memory Refresh Overflow Error	13-8
13.3.3	PCI Interface	13-8
13.3.3.1	Address Parity Error	13-8
13.3.3.2	Data Parity Error	13-8
13.3.3.3	Master-Abort Transaction Termination	13-9
13.3.3.4	Received Target-Abort Error	13-9
13.3.3.5	NMI (Nonmaskable Interrupt)	13-9
13.4	Exception Latencies	13-10

CONTENTS

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

Chapter 14 Power Management

14.1	Overview	14-1
14.2	Processor Core Power Management	14-1
14.2.1	Dynamic Power Management	14-2
14.2.2	Programmable Power Modes on Processor Core	14-2
14.2.3	Processor Power Management Modes—Details	14-3
14.2.3.1	Full-Power Mode with DPM Disabled	14-4
14.2.3.2	Full-Power Mode with DPM Enabled	14-4
14.2.3.3	Processor Doze Mode	14-4
14.2.3.4	Processor Nap Mode	14-5
14.2.3.5	Processor Sleep Mode	14-5
14.2.4	Power Management Software Considerations	14-6
14.3	Peripheral Logic Power Management	14-7
14.3.1	Peripheral Logic Power Modes	14-7
14.3.1.1	Peripheral Logic Full Power Mode	14-8
14.3.1.2	Peripheral Logic Doze Mode	14-8
14.3.1.3	Peripheral Logic Nap Mode	14-9
14.3.1.3.1	PCI Transactions During Nap Mode	14-9
14.3.1.3.2	PLL Operation During Nap Mode	14-9
14.3.1.4	Peripheral Logic Sleep Mode	14-9
14.3.1.4.1	System Memory Refresh during Sleep Mode	14-10
14.3.1.4.2	Disabling the PLL during Sleep Mode	14-10
14.3.1.4.3	PCI Transactions in Sleep Mode	14-10
14.3.1.4.4	SDRAM Paging During Sleep Mode	14-10
14.4	Example Code Sequence for Entering Processor Sleep Mode	14-11

Chapter 15 Debug Features

15.1	Address Attribute Signals	15-1
15.1.1	Memory Address Attribute Signals (MAA[0–2])	15-1
15.1.2	Memory Address Attribute Signal Timing	15-2
15.1.3	PCI Address Attribute Signals	15-2
15.1.4	PCI Address Attribute Signal Timing	15-3
15.2	Memory Debug Address	15-4
15.2.1	Enabling Debug Address	15-4
15.2.2	Debug Address Signal Definitions	15-4
15.2.3	Physical Address Mappings	15-5
15.2.4	RAS/CS Encoding	15-6
15.2.5	Debug Address Timing	15-7

CONTENTS

Paragraph Number	Title	Page Number
15.3	Memory Interface Valid (\overline{MIV}).....	15-7
15.3.1	\overline{MIV} Signal Timing	15-8
15.4	Memory Datapath Error Injection/Capture.....	15-16
15.4.1	Memory Data Path Diagnostic Registers Address Map.....	15-16
15.4.2	Memory Data Path Error Injection Mask Registers	15-17
15.4.2.1	Data High Error Injection Mask Register.....	15-18
15.4.2.2	Data Low Error Injection Mask Register	15-18
15.4.2.3	Parity Error Injection Mask Register.....	15-19
15.4.3	Memory Data Path Error Capture Monitor Registers.....	15-19
15.4.3.1	Data High Error Capture Monitor Register	15-20
15.4.3.2	Data Low Error Capture Monitor Register.....	15-20
15.4.3.3	Parity Error Capture Monitor Register	15-20
15.5	JTAG/Testing Support.....	15-21
15.5.1	JTAG Signals.....	15-22
15.5.2	JTAG Registers and Scan Chains.....	15-22
15.5.2.1	Bypass Register	15-22
15.5.2.2	Boundary-Scan Registers	15-23
15.5.2.3	Instruction Register	15-23
15.5.2.4	TAP Controller.....	15-23

Appendix A PowerPC Instruction Set Listings

A.1	Instructions Sorted by Mnemonic.....	A-1
A.2	Instructions Sorted by Opcode	A-9
A.3	Instructions Grouped by Functional Categories	A-17
A.4	Instructions Sorted by Form	A-27
A.5	Instruction Set Legend.....	A-38

Appendix B Bit and Byte Ordering

B.1	Byte Ordering Overview	B-1
B.2	Byte-Ordering Mechanisms.....	B-1
B.3	Big-Endian Mode	B-2
B.4	Little-Endian Mode.....	B-6
B.4.1	I/O Addressing in Little-Endian Mode.....	B-15
B.5	Setting the Endian Mode Of Operation.....	B-15

ILLUSTRATIONS

Figure Number	Title	Page Number
1-1	MPC8240 Integrated Processor Functional Block Diagram.....	1-2
1-2	System Using an Integrated MPC8240 as a Host Processor.....	1-5
1-3	Embedded System Using an MPC8240 as a Distributed Processor	1-6
1-4	Embedded System Using an MPC8240 as a Peripheral Processor.....	1-7
1-5	MPC8240 Integrated Processor Core Block Diagram.....	1-9
1-6	MPC8240 Peripheral Logic Block Diagram.....	1-11
2-1	MPC8240 Integrated Processor Core Block Diagram.....	2-2
2-2	MPC8240 Programming Model—Registers.....	2-12
2-3	Hardware Implementation Register 0 (HID0)	2-13
2-4	Hardware Implementation Register 1 (HID1)	2-17
2-5	Hardware Implementation-Dependent Register 2 (HID2).....	2-17
2-6	Data Cache Organization	2-22
3-1	MPC8240 Signal Groupings.....	3-3
3-2	Clock Subsystem Block Diagram	3-32
3-3	Timing Diagram (1X, 1.5X, 2X, 2.5X, and 3X examples).....	3-33
3-4	Clocking Solution—Small Load Requirements.....	3-35
3-5	Clocking Solution—High Clock Fanout Required	3-35
4-1	Processor Core Address Map A	4-3
4-2	PCI Master Memory Address Map A	4-4
4-3	PCI Master I/O Address Map A.....	4-5
4-4	Address Map B in Host Mode	4-9
4-5	Inbound PCI Address Translation.....	4-10
4-6	Outbound PCI Address Translation	4-12
4-7	Local Memory Base Address Register (LMBAR)—0x10.....	4-13
4-8	Inbound Translation Window Register (ITWR).....	4-14
4-9	Outbound Memory Base Address Register (OMBAR)—0x0_2300	4-15
4-10	Outbound Translation Window Register (OTWR)—0x0_2308.....	4-16
4-11	Embedded Utilities Memory Block Mapping to Local Memory.....	4-17
4-12	Embedded Utilities Memory Block Mapping to PCI Memory.....	4-18
5-1	Processor Accessible Configuration Space.....	5-6
5-2	PCI Accessible Configuration Space	5-7
5-3	PCI Command Register	5-10
5-4	PCI Status Register	5-11
5-5	Power Management Configuration Register 1 (PMCR1).....	5-16
5-6	Power Management Configuration Register 2 (PMCR2).....	5-17
5-7	Processor Interface Configuration Register 1 (PICR1)—0xA8.....	5-22

ILLUSTRATIONS

Figure Number	Title	Page Number
5-8	Processor Interface Configuration Register 2 (PICR2)—0xAC	5-24
5-9	ECC Single-Bit Error Counter Register—0xB8	5-26
5-10	ECC Single-Bit Error Trigger Register—0xB9	5-27
5-11	Error Enabling Register 1 (ErrEnR1)—0xC0	5-27
5-12	Error Enabling Register 2 (ErrEnR2)—0xC4	5-28
5-13	Error Detection Register 1 (ErrDR1)—0xC1	5-30
5-14	Error Detection Register 2 (ErrDR2)—0xC5	5-31
5-15	Internal Processor Bus Error Status Register—0xC3	5-31
5-16	PCI Bus Error Status Register—0xC7	5-32
5-17	Processor/PCI Error Address Register—0xC8	5-32
5-18	Address Map B Options Register (AMBOR)—0xE0	5-33
5-19	Memory Starting Address Register 1—0x80	5-35
5-20	Memory Starting Address Register 2—0x84	5-35
5-21	Extended Memory Starting Address Register 1—0x88	5-36
5-22	Extended Memory Starting Address Register 2—0x8C	5-36
5-23	Memory Ending Address Register 1—0x90	5-37
5-24	Memory Ending Address Register 2—0x94	5-37
5-25	Extended Memory Ending Address Register 1—0x98	5-38
5-26	Extended Memory Ending Address Register 2—0x9C	5-38
5-27	Memory Bank Enable Register—0xA0	5-39
5-28	Memory Page Mode Register—0xA3	5-40
5-29	Memory Control Configuration Register 1 (MCCR1)—0xF0	5-41
5-30	Memory Control Configuration Register 2 (MCCR2)—0xF4	5-44
5-31	Memory Control Configuration Register 3 (MCCR3)—0xF8	5-46
5-32	Memory Control Configuration Register 4 (MCCR4)—0xFC	5-49
6-1	Block Diagram for Memory Interface	6-3
6-2	FPM or EDO DRAM Memory Interface Block Diagram	6-6
6-3	Example 16-Mbyte DRAM System with Parity—64-Bit Mode	6-8
6-4	DRAM Memory Organization	6-9
6-5	DRAM Address Multiplexing SDMA[12-0]—32 Bit Mode	6-13
6-6	DRAM Address Multiplexing SDMA[12-0]—64 Bit Mode	6-14
6-7	FPM-EDO Flow-through Memory Interface	6-16
6-8	DRAM Single-Beat Read Timing (No ECC)	6-19
6-9	DRAM Four-Beat Burst Read Timing (No ECC)—64-Bit Mode	6-19
6-10	DRAM Eight-Beat Burst Read Timing Configuration—32-Bit Mode	6-20
6-11	DRAM Single-Beat Write Timing (No ECC)	6-21
6-12	DRAM Four-Beat Burst Write Timing (No ECC)—64-Bit Mode	6-21
6-13	DRAM Eight-beat Burst Write Timing (No ECC)—32 Bit Mode	6-22
6-14	FPM DRAM Burst Read with ECC	6-27
6-15	EDO DRAM Burst Read Timing with ECC	6-27
6-16	DRAM Single-Beat Write Timing with RMW or ECC Enabled	6-28
6-17	DRAM Bank Staggered CBR Refresh Timing Configuration	6-29
6-18	DRAM Self-Refresh Timing Configuration	6-30

ILLUSTRATIONS

Figure Number	Title	Page Number
6-19	SDRAM Memory Interface Block Diagram.....	6-31
6-20	SDRAM Data Bus Lane Assignments.....	6-32
6-21	Example 128 Mbyte SDRAM Configuration With Parity.....	6-34
6-22	SDRAM Address Multiplexing SDBA[1-0] and SDMA[12-0]—32-Bit Mode	6-37
6-23	SDRAM Address Multiplexing SDBA[1-0] and SDMA[12-0]—64-Bit Mode	6-38
6-24	SDRAM Flow-Through Memory Interface	6-40
6-25	SDRAM Registered Memory Interface	6-41
6-26	SDRAM In-line ECC/Parity Memory Interface	6-41
6-27	PGMAX Parameter Setting for SDRAM Interface	6-45
6-28	SDRAM Single-Beat Read Timing (SDRAM Burst Length = 4)	6-50
6-29	SDRAM Four-Beat Burst Read Timing Configuration—64-Bit Mode	6-51
6-30	SDRAM Eight-Beat Burst Read Timing Configuration—32-Bit Mode	6-51
6-31	SDRAM Single Beat Write Timing (SDRAM Burst Length = 4).....	6-52
6-32	SDRAM Four-Beat Burst Write Timing—64-Bit Mode	6-52
6-33	SDRAM Eight-Beat Burst Write Timing—32-Bit Mode.....	6-53
6-34	SDRAM Mode Register Set Timing.....	6-53
6-35	Registered SDRAM DIMM Single Beat Write Timing	6-55
6-36	Registered SDRAM DIMM Burst-Write Timing	6-56
6-37	SDRAM Bank Staggered CBR Refresh Timing.....	6-58
6-38	SDRAM Self Refresh Entry.....	6-60
6-39	SDRAM Self Refresh Exit.....	6-60
6-40	ROM Memory Interface Block Diagram.....	6-61
6-41	16-Mbyte ROM System Including Parity Paths to DRAM—64-Bit Mode.....	6-62
6-42	1-Mbyte Flash Memory System Including Parity Paths to DRAM—8-Bit Mode	6-63
6-43	ROM/Flash Address Multiplexing—8-Bit Mode	6-65
6-44	ROM/Flash Address Multiplexing—32-Bit Mode	6-66
6-45	ROM/Flash Address Multiplexing—64-Bit Mode	6-66
6-46	Non Burst ROM/Flash Read Access Timing—32- or 64-Bit Mode.....	6-68
6-47	Burst ROM/Flash Read Access Timing (Cache Block)—64-Bit Mode.....	6-68
6-48	Burst ROM/Flash Read Access Timing (Cache Block)—32-Bit Mode.....	6-68
6-49	8-Bit ROM/Flash Read Access Timing	6-70
6-50	8-, 32-, or 64-Bit Flash Write Access Timing	6-71
6-51	Port X Peripheral Interface Block Diagram.....	6-72
6-52	Example of Port X Peripheral Connected to the MPC8240	6-74
6-53	Example of Port X Peripheral Connected to the MPC8240	6-74
6-54	Port X Example Read Access Timing	6-75
6-55	Port X Example Write Access Timing.....	6-75
7-1	MPC8240 Internal Buffer Organization	7-2
7-2	Processor/System Memory Buffers	7-3
7-3	Processor/PCI Buffers.....	7-4

ILLUSTRATIONS

Figure Number	Title	Page Number
7-4	PCI/System Memory Buffers.....	7-6
8-1	PCI Arbitration Example	8-5
8-2	PCI Single-Beat Read Transaction	8-13
8-3	PCI Burst Read Transaction.....	8-13
8-4	PCI Single-Beat Write Transaction	8-14
8-5	PCI Burst Write Transaction.....	8-14
8-6	PCI Target-Initiated Terminations.....	8-17
8-7	Standard PCI Configuration Header	8-18
8-8	Layout of CONFIG_ADDR Register	8-20
8-9	Type 0 Configuration Translation.....	8-22
8-10	Direct-Access PCI Configuration Transaction	8-24
8-11	PCI Parity Operation.....	8-29
9-1	DMA Controller Block Diagram	9-2
9-2	Chaining of DMA Descriptors in Memory.....	9-10
9-3	DMA Mode Register (DMR).....	9-12
9-4	DMA Status Register (DSR).....	9-14
9-5	Current Descriptor Address Register (CDAR)	9-16
9-6	Source Address Register (SAR).....	9-17
9-7	Destination Address Register (DAR).....	9-17
9-8	Byte Count Register (BCR)	9-18
9-9	Next Descriptor Address Register (NDAR)	9-18
10-1	Message Registers (IMRs and OMRs)	10-2
10-2	Inbound Doorbell Register (IDBR)	10-3
10-3	Outbound Doorbell Register (ODBR)	10-3
10-4	I ₂ O Message Queue Example	10-6
10-5	Outbound Message Interrupt Status Register (OMISR)	10-9
10-6	Outbound Message Interrupt Mask Register (OMIMR).....	10-10
10-7	Inbound FIFO Queue Port Register (IFQPR)	10-10
10-8	Outbound FIFO Queue Port Register (OFQPR).....	10-11
10-9	Inbound Message Interrupt Status Register (IMISR)	10-12
10-10	Inbound Message Interrupt Mask Register (IMIMR)	10-13
10-11	Inbound Free_FIFO Head Pointer Register (IFHPR)	10-14
10-12	Inbound Free_FIFO Tail Pointer Register (IFTPR).....	10-15
10-13	Inbound Post_FIFO Head Pointer Register (IPHPR)	10-16
10-14	Inbound Post_FIFO Tail Pointer Register (IPTPR).....	10-16
10-15	Outbound Free_FIFO Head Pointer Register (OFHPR).....	10-17
10-16	Outbound Free_FIFO Tail Pointer Register (OFTPR)	10-17
10-17	Outbound Post_FIFO Head Pointer Register (OPHPR)	10-18
10-18	Outbound Post_FIFO Tail Pointer Register (OPTPR).....	10-19
10-19	Messaging Unit Control Register (MUCR)	10-19
10-20	Queue Base Address Register (QBAR)	10-20
11-1	I ² C Interface Block Diagram	11-3
11-2	I ² C Interface Transaction Protocol	11-4

ILLUSTRATIONS

Figure Number	Title	Page Number
11-3	I ² C Address Register (I2CADR)	11-8
11-4	I ² C Frequency Divider Register (I2CFDR)	11-8
11-5	I ² C Control Register (I2CCR)	11-10
11-6	I ² C Status Register (I2CSR)	11-12
11-7	I ² C Data Register (I2CDR)	11-13
11-8	Example I ² C Interrupt Service Routine Flowchart	11-18
12-1	EPIC Unit Block Diagram	12-3
12-2	EPIC Interrupt Generation Block Diagram— Non-programmable Registers	12-8
12-3	Serial Interrupt Interface Protocol	12-12
12-4	Feature Reporting Register (FRR)	12-15
12-5	Global Configuration Register (GCR)	12-15
12-6	EPIC Interrupt Configuration Register (EICR)	12-16
12-7	EPIC Vendor Identification Register (EVI)	12-17
12-8	Processor Initialization Register (PI)	12-18
12-9	Spurious Vector Register (SVR)	12-18
12-10	Timer Frequency Reporting Register (TFRR)	12-19
12-11	Global Timer Current Count Register (GTCCR)	12-20
12-12	Global Timer Base Count Register (GTBCR)	12-20
12-13	Global Timer Vector/Priority Register (GTVPR)	12-21
12-14	Global Timer Destination Register (GTDR)	12-22
12-15	Direct and Serial Interrupt Vector/Priority Registers (IVPR and SVPR)	12-23
12-16	Direct and Serial Destination Registers (IDR and SDR)	12-24
12-17	Processor Current Task Priority Register (PCTPR)	12-25
12-18	Processor Interrupt Acknowledge Register (IACK)	12-26
12-19	Processor End of Interrupt Register (EOI)	12-26
13-1	Internal Error Management Block Diagram	13-2
14-1	MPC8240 Peripheral Logic Power States	14-7
15-1	Example PCI Address Attribute Signal Timing for Burst Read Operations	15-3
15-2	Example PCI Address Attribute Signal Timing for Burst Write Operations	15-4
15-3	64-Bit Mode, DRAM and SDRAM Physical Address for Debug	15-5
15-4	32-Bit Mode, DRAM and SDRAM Physical Address for Debug	15-5
15-5	64-Bit Mode, ROM and Flash Physical Address for Debug	15-6
15-6	32-Bit Mode, ROM and Flash Physical Address for Debug	15-6
15-7	8-bit mode, ROM and FLASH Physical Address for Debug	15-6
15-8	Example FPM Debug Address, MIV, and MAA Timings for Burst Read Operation	15-8
15-9	Example FPM Debug Address, MIV, and MAA Timings for Burst Write Operation	15-9
15-10	Example EDO Debug Address, MIV, and MAA Timings for Burst Read Operation	15-10
15-11	Example EDO Debug Address, MIV, and MAA Timings for Burst Write Operation	15-11

ILLUSTRATIONS

Figure Number	Title	Page Number
15-12	Example SDRAM Debug Address, MIV, and MAA Timings for Burst Read Operation.....	15-12
15-13	Example SDRAM Debug Address, MIV, and MAA Timings for Burst Write Operation.....	15-13
15-14	Example ROM Debug Address, MIV, and MAA Timings For Burst Read....	15-14
15-15	Example Flash Debug Address, MIV, and MAA Timings For Single-Byte Read	15-15
15-16	Example Flash Debug Address, MIV, and MAA Timings for Write Operation.....	15-16
15-17	Functional Diagram of Memory Data Path Error Injection	15-17
15-18	Data High Error Injection Mask (MDP_ERR_INJ_MASK_DH): 4Bytes @ <FF00, F00>	15-18
15-19	Data Low Error Injection Mask (MDP_ERR_INJ_MASK_DL): 4Bytes @ <FF04, F04>	15-18
15-20	Parity Error Injection Mask (MDP_ERR_INJ_MASK_PAR): 4Bytes @ <FF08, F08>	15-19
15-21	Data High Error Capture Monitor (MDP_ERR_CAP_MON_DH): 4Bytes @ <FF0c, F0c>.....	15-20
15-22	Data Low Error Capture Monitor (MDP_ERR_CAP_MON_DL): 4Bytes @ <FF10, F10>	15-20
15-23	Parity Error Capture Monitor (MDP_ERR_CAP_MON_PAR): 4Bytes @ <FF14, F14>	15-21
15-24	JTAG Interface Block Diagram	15-22
B-1	Four-Byte Transfer to PCI Memory Space—Big-Endian Mode.....	B-3
B-2	Big-Endian Memory Image in Local Memory	B-4
B-3	Big-Endian Memory Image in Big-Endian PCI Memory Space.....	B-5
B-4	Munged Memory Image in local memory.....	B-7
B-5	Little-Endian Memory Image in Little-Endian PCI Memory Space	B-8
B-6	One-Byte Transfer to PCI Memory Space—Little-Endian Mode.....	B-9
B-7	Two-Byte Transfer to PCI Memory Space—Little-Endian Mode	B-10
B-8	Four-Byte Transfer to PCI Memory Space—Little-Endian Mode.....	B-11
B-9	One-Byte Transfer to PCI I/O Space—Little-Endian Mode	B-12
B-10	Two-Byte Transfer to PCI I/O Space—Little-Endian Mode	B-13
B-11	Four-Byte Transfer to PCI I/O Space—Little-Endian Mod	B-14

TABLES

Table Number	Title	Page Number
i	Acronyms and Abbreviated Terms	xxxviii
1-1	Programmable Processor Power Modes	1-18
1-2	Programmable Peripheral Logic Power Modes	1-18
2-1	HID0 Field Descriptions	2-13
2-2	HID0[BCLK] and HID0[ECLK] CKO Signal Configuration	2-16
2-3	HID1 Field Descriptions	2-17
2-4	HID2 Field Descriptions	2-17
2-5	CCU Responses to Processor Transactions	2-24
2-6	Transactions Reflected to the Processor for Snooping	2-26
2-7	Exception Classifications for the Processor Core	2-28
2-8	Exceptions and Conditions	2-28
2-9	Integer Divide Latency	2-34
2-10	Major Differences between MPC8240's Core and the MPC603e User's Manual.....	2-35
3-1	MPC8240 Signal Cross Reference.....	3-4
3-2	Output Signal States During System Reset.....	3-7
3-3	PCI Command Encodings.....	3-8
3-4	Data Bus Byte Lane Assignments.....	3-19
3-5	Reset Configuration Signals	3-36
4-1	Address Map A—Processor View	4-2
4-2	Address Map A—PCI Memory Master View	4-2
4-3	Address Map A—PCI I/O Master View	4-2
4-4	Address Map B—Processor View	4-6
4-5	Address Map B—PCI Memory Master View (Host Bridge).....	4-6
4-6	Address Map B—PCI Memory Master View (Agent Bridge)	4-7
4-7	Address Map B—PCI I/O Master View	4-8
4-8	ATU Register Summary	4-13
4-9	Bit Settings for LMBAR—0x10.....	4-13
4-10	Bit Settings for ITWR—0x0_2310.....	4-14
4-11	Bit Settings for OMBAR—0x0_2300	4-15
4-12	Bit Settings for OTWR—0x0_2308	4-16
4-13	Embedded Utilities Local Memory Register Summary	4-18
4-14	Embedded Utilities Peripheral Control and Status Register Summary	4-19
5-1	MPC8240 Configuration Registers Accessible from the Processor Core	5-4
5-2	MPC8240 Configuration Registers Accessible from the PCI Bus	5-7
5-3	PCI Configuration Space Header Summary	5-8

TABLES

Table Number	Title	Page Number
5-4	Bit Settings for PCI Command Register—0x04.....	5-10
5-5	Bit Settings for PCI Status Register—0x06.....	5-12
5-6	Programming Interface—0x09.....	5-12
5-7	PCI Base Class Code—0x0B.....	5-13
5-8	Cache Line Size Register—0x0C.....	5-13
5-9	Latency Timer Register—0x0D.....	5-13
5-10	Local Memory Base Address Register Bit Definitions—0x10.....	5-14
5-11	PCSR Base Address Register Bit Definitions—0x14.....	5-14
5-12	Interrupt Line Register—0x3C.....	5-14
5-13	PCI Arbiter Control Register Bit Definitions—0x46.....	5-15
5-14	Bit Settings for Power Management Configuration Register 1—0x70.....	5-16
5-15	Power Management Configuration Register 2—0x72.....	5-18
5-16	Output Driver Control Register Bit Definitions—0x73.....	5-19
5-17	CLK Driver Control Register Bit Definitions—0x74.....	5-20
5-18	Embedded Utilities Memory Base Address Register—0x78.....	5-21
5-19	Bit Settings for PICR1—0xA8.....	5-22
5-20	Bit Settings for PICR2—0xAC.....	5-24
5-21	Bit Settings for ECC Single-Bit Error Counter Register—0xB8.....	5-26
5-22	Bit Settings for ECC Single-Bit Error Trigger Register—0xB9.....	5-27
5-23	Bit Settings for Error Enabling Register 1 (ErrEnR1)—0xC0.....	5-28
5-24	Bit Settings for Error Enabling Register 2 (ErrEnR2)—0xC4.....	5-29
5-25	Bit Settings for Error Detection Register 1 (ErrDR1)—0xC1.....	5-30
5-26	Bit Settings for Error Detection Register 2 (ErrDR2)—0xC5.....	5-31
5-27	Bit Settings for Internal Processor Bus Error Status Register—0xC3.....	5-32
5-28	Bit Settings for PCI Bus Error Status Register—0xC7.....	5-32
5-29	Bit Settings for Processor/PCI Error Address Register—0xC8.....	5-33
5-30	Bit Settings for the AMBOR—0xE0.....	5-34
5-31	Bit Settings for Memory Starting Address Registers 1 and 2.....	5-35
5-32	Bit Settings for Extended Memory Starting Address Registers 1 and 2.....	5-36
5-33	Bit Settings for Memory Ending Address Registers 1 and 2.....	5-37
5-34	Bit Settings for Extended Memory Ending Address Registers 1 and 2.....	5-38
5-35	Bit Settings for Memory Bank Enable Register—0xA0.....	5-39
5-36	Bit Settings for Memory Page Mode Register—0xA3.....	5-40
5-37	Bit Settings for MCCR1—0xF0.....	5-42
5-38	Bit Settings for MCCR2—0xF4.....	5-44
5-39	Bit Settings for MCCR3—0xF8.....	5-47
5-40	Bit Settings for MCCR4—0xFC.....	5-50
6-1	Memory Interface Signal Summary.....	6-4
6-2	Memory Address Signal Mappings.....	6-5
6-3	Unsupported Multiplexed Row and Column Address Bits.....	6-10
6-4	Supported FPM or EDO DRAM Device Configurations.....	6-10
6-5	SDMA[11–8] Encodings for 32- and 64-Bit Bus Modes.....	6-12
6-6	FPM or EDO Memory Parameters.....	6-15

TABLES

Table Number	Title	Page Number
6-7	FPM or EDO System Configurations	6-15
6-8	Memory Interface Configuration Register Fields	6-16
6-9	FPM or EDO Timing Parameters	6-18
6-10	The MPC8240 FPM or EDO ECC Syndrome Encoding (Data bits 0–31).....	6-25
6-11	The MPC8240 FPM or EDO ECC Syndrome Encoding (Data bits 32–63).....	6-25
6-12	FPM or EDO DRAM Power Saving Modes Refresh Configuration	6-29
6-13	Unsupported Multiplexed Row and Column Address Bits.....	6-35
6-14	Supported SDRAM Device Configurations—64-bit Mode.....	6-36
6-15	Bit Name and Location Cross-listing.....	6-39
6-16	SDRAM System Configurations.....	6-40
6-17	The MPC8240 SDRAM ECC Syndrome Encoding (Data Bits 0–31)	6-43
6-18	The MPC8240 SDRAM ECC Syndrome Encoding (Data Bits 32–63)	6-43
6-19	MPC8240 SDRAM Interface Command and Data Inputs.....	6-49
6-20	SDRAM Interface Timing Intervals	6-50
6-21	SDRAM Controller Power Saving Configurations.....	6-59
6-22	SDRAM Power Saving Modes Refresh Configuration	6-59
6-23	Reset Configurations of ROM/Flash Controller	6-64
7-1	Snooping Behavior Caused by a Hit in an Internal Buffer	7-7
7-2	Internal Arbitration Priorities.....	7-9
8-1	PCI Bus Commands.....	8-8
8-2	PCI Configuration Space Header Summary	8-19
8-3	CONFIG_ADDR Register Fields	8-20
8-4	Type 0 Configuration—Device Number to IDSEL Translation.....	8-22
8-5	Special-Cycle Message Encodings	8-26
8-6	Initialization Options for PCI Controller	8-30
9-1	DMA Register Summary	9-2
9-2	DMA Descriptor Summary.....	9-9
9-3	DMR Field Descriptions—0x100	9-12
9-4	DSR Field Descriptions—0x104	9-15
9-5	CDAR Field Descriptions—0x108	9-16
9-6	SAR Field Description—0x110	9-17
9-7	DAR Field Description—0x118	9-17
9-8	BCR Field Descriptions—0x120	9-18
9-9	NDAR Field Descriptions—0x124.....	9-19
10-1	Message Register Summary.....	10-2
10-2	Doorbell Register Summary	10-2
10-3	IMR and OMR Field Descriptions— Offset 0x050–0x05C	10-2
10-4	IDBR Field Descriptions— Offset 0x068.....	10-3
10-5	ODBR Field Descriptions— Offset 0x060	10-4
10-6	I ₂ O PCI Configuration Identification Register Settings	10-4
10-7	I ₂ O Register Summary.....	10-5
10-8	Queue Starting Address	10-6
10-9	OMISR Field Descriptions— Offset 0x030	10-9

TABLES

Table Number	Title	Page Number
10-10	OMIMR Field Descriptions— Offset 0x034	10-10
10-11	IFQPR Field Descriptions— Offset 0x040.....	10-11
10-12	OFQPR Field Descriptions— Offset 0x044	10-11
10-13	IMISR Field Descriptions— Offset 0x0_0100	10-12
10-14	IMIMR Field Descriptions— Offset 0x0_0104.....	10-14
10-15	IFHPR Field Descriptions— Offset 0x0_0120	10-15
10-16	IFTPR Field Descriptions— Offset 0x0_0128	10-15
10-17	IPHPR Field Descriptions— Offset 0x0_0130	10-16
10-18	IPTPR Field Descriptions— Offset 0x0_0138	10-16
10-19	OFHPR Field Descriptions— Offset 0x0_0140	10-17
10-20	OFTPR Field Descriptions— Offset 0x0_0148.....	10-18
10-21	OPHPR Field Descriptions— Offset 0x0_0150	10-18
10-22	OPTPR Field Descriptions— Offset 0x0_0158.....	10-19
10-23	MUCR Field Descriptions— Offset 0x0_0164	10-20
10-24	QBAR Field Descriptions— Offset 0x0_0170	10-20
11-1	I ² C Interface Signal Description.....	11-2
11-2	I ² C Register Summary	11-7
11-3	I2CADR Field Descriptions— Offset 0x0_3000.....	11-8
11-4	I2CFDR Field Descriptions— Offset 0x0_3004	11-9
11-5	Serial Bit Clock Frequency Divider Selections	11-9
11-6	I2CCR Field Descriptions— Offset 0x0_3008.....	11-11
11-7	I2CSR Field Descriptions— Offset 0x0_300C.....	11-12
11-8	I2CDR Field Descriptions— Offset 0x0_3010.....	11-14
12-1	EPIC Interface Signal Description.....	12-2
12-2	EPIC Register Address Map— Global and Timer Registers.....	12-4
12-3	EPIC Register Address Map— Interrupt Source Configuration Registers.....	12-5
12-4	EPIC Register Address Map— Processor-Related Registers	12-6
12-5	FRR Field Descriptions— Offset 0x4_1000.....	12-15
12-6	GCR Field Descriptions— Offset 0x4_1020	12-16
12-7	EICR Field Descriptions— Offset 0x4_1030	12-17
12-8	EVI Register Field Descriptions— Offset 0x4_1080	12-17
12-9	PI Register Field Descriptions— Offset 0x4_1090	12-18
12-10	SVR Field Descriptions— Offset 0x4_10E0.....	12-19
12-11	TFRR Field Descriptions— Offset 0x4_10F0.....	12-19
12-12	GTCCR Field Descriptions.....	12-20
12-13	GTBCR Field Descriptions.....	12-21
12-14	GTVPR Field Descriptions	12-21
12-15	GTDR Field Descriptions	12-22
12-16	IVPR and SVPR Field Descriptions	12-23
12-17	IDR and SDR Field Descriptions.....	12-24
12-18	PCTPR Field Descriptions— Offset 0x6_0080.....	12-25
12-19	IACK Field Descriptions— Offset 0x6_00A0	12-26
12-20	EOI Field Descriptions— Offset 0x6_00B0.....	12-26

TABLES

Table Number	Title	Page Number
13-1	MPC8240 Error Priorities	13-2
14-1	Programmable Processor Power Modes	14-3
14-2	Programmable Peripheral Logic Power Modes	14-8
15-1	Address Attribute Signal Summary	15-1
15-2	Memory Address Attribute Signal Encodings	15-2
15-3	PCI Attribute Signal Encodings	15-2
15-4	Memory Debug Address Signal Definitions	15-5
15-5	Example of Chip Select Encoding For 568 Mbyte Memory System	15-6
15-6	Memory Interface Valid Signal Definition	15-8
15-7	Memory Data Path Diagnostic Register Offsets	15-17
15-8	Data High Error Injection Mask Bit Field Definitions	15-18
15-9	Data Low Error Injection Mask Bit Field Definitions	15-18
15-10	Parity Error Injection Mask Bit Field Definitions	15-19
15-11	Data High Error Capture Monitor Bit Field Definitions	15-20
15-12	Data Low Error Capture Monitor Bit Field Definitions	15-20
15-13	Parity High Error Capture Monitor Bit Field Definitions	15-21
A-1	Complete Instruction List Sorted by Mnemonic	A-1
A-2	Complete Instruction List Sorted by Opcode	A-9
A-3	Integer Arithmetic Instructions	A-17
A-4	Integer Compare Instructions	A-18
A-5	Integer Logical Instructions	A-18
A-6	Integer Rotate Instructions	A-18
A-7	Integer Shift Instructions	A-19
A-8	Floating-Point Arithmetic Instructions ⁷	A-19
A-9	Floating-Point Multiply-Add Instructions ⁷	A-20
A-10	Floating-Point Rounding and Conversion Instructions ⁷	A-20
A-11	Floating-Point Compare Instructions ⁷	A-20
A-12	Floating-Point Status and Control Register Instructions ⁷	A-20
A-13	Integer Load Instructions	A-21
A-14	Integer Store Instructions	A-21
A-15	Integer Load and Store with Byte-Reverse Instructions	A-22
A-16	Integer Load and Store Multiple Instructions	A-22
A-17	Integer Load and Store String Instructions	A-22
A-18	Memory Synchronization Instructions	A-22
A-19	Floating-Point Load Instructions ⁷	A-23
A-20	Floating-Point Store Instructions ⁷	A-23
A-21	Floating-Point Move Instructions ⁷	A-23
A-22	Branch Instructions	A-24
A-23	Condition Register Logical Instructions	A-24
A-24	System Linkage Instructions	A-24
A-25	Trap Instructions	A-24
A-26	Processor Control Instructions	A-24
A-27	Cache Management Instructions	A-25

TABLES

Table Number	Title	Page Number
A-28	Segment Register Manipulation Instructions	A-25
A-29	Lookaside Buffer Management Instructions	A-25
A-30	External Control Instructions	A-26
A-31	I-Form.....	A-27
A-32	B-Form	A-27
A-33	SC-Form	A-27
A-34	D-Form	A-27
A-35	DS-Form	A-29
A-36	X-Form	A-29
A-37	XL-Form.....	A-33
A-38	XFX-Form	A-34
A-39	XFL-Form.....	A-34
A-40	XS-Form	A-34
A-41	XO-Form	A-34
A-42	A-Form	A-35
A-43	M-Form.....	A-36
A-44	MD-Form.....	A-36
A-45	MDS-Form	A-37
A-46	PowerPC Instruction Set Legend.....	A-38
B-1	Byte Lane Translation in Big-Endian Mode	B-2
B-2	Processor Address Modification for Individual Aligned Scalars	B-6
B-3	MPC8240 Address Modification for Individual Aligned Scalars	B-6
B-4	Byte Lane Translation in Little-Endian Mode.....	B-7

About This Book

The primary objective of this user's manual is to define the functionality of the MPC8240 PowerPC™ integrated processor. The MPC8240 has a processor core based on the PowerPC 603e™ low-power microprocessor; it also performs many peripheral functions on-chip.

The MPC603e implements the full 32-bit portion of the PowerPC™ architecture. It is important to note that this book is intended as a companion to the following publications:

- *The MPC603e & EC603e User's Manual*
- *The PowerPC Microprocessor Family: The Programming Environments*, (referred to as *The Programming Environments Manual*)

Contact your local sales representative to obtain a copy. Because the PowerPC architecture is designed to support a broad range of processors, *The Programming Environments Manual* provides a general description of features that are common to PowerPC processors and indicates those features that are optional or may be implemented differently in the design of each processor.

The information in this book is subject to change without notice, as described in the disclaimers on the title page of this book. As with any technical documentation, it is the readers' responsibility to use the most recent version of the documentation. For more information, contact your sales representative.

Audience

This manual is intended for system software and hardware developers and applications programmers who want to develop products using the MPC8240 integrated processor. It is assumed that the reader understands operating systems, microprocessor system design, the basic principles of RISC processing, and details of the PowerPC architecture.

Organization

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

- Chapter 1, "Overview," is useful for readers who want a general understanding of the features and functions of the MPC8240 device and its component parts.
- Chapter 2, "PowerPC Processor Core," provides an overview of the basic functionality of the G2 processor core.

- Chapter 3, “Signal Descriptions and Clocking,” provides descriptions of the MPC8240’s external signals. It describes each signal’s behavior when the signal is asserted and negated and when the signal is an input or an output.
- Chapter 4, “Address Maps,” describes how the MPC8240 in host mode supports two address mapping configurations—address map A and address map B.
- Chapter 5, “Configuration Registers,” describes the programmable configuration registers of the MPC8240.
- Chapter 6, “MPC8240 Memory Interface,” describes the memory interface of the MPC8240 and how it controls the processor and PCI interactions to main memory.
- Chapter 7, “Central Control Unit,” describes the internal buffering and arbitration logic of the MPC8240 central control unit (CCU).
- Chapter 8, “PCI Bus Interface,” provides a rudimentary description of PCI bus operations. The specific emphasis is directed at how the MPC8240 implements the PCI bus.
- Chapter 9, “DMA Controller,” describes how the DMA controller operates on the MPC8240.
- Chapter 10, “Message Unit (with I₂O),” describes a mechanism to facilitate communications between host and peripheral processors.
- Chapter 11, “I²C Interface,” describes a simple, efficient method of data exchange between IC devices. The I²C interface allows the MPC8240 to exchange data with other I²C devices such as microcontrollers, EEPROMs, real-time clock devices, A/D converters, and LCDs.
- Chapter 12, “Embedded Programmable Interrupt Controller (EPIC),” provides a description of a general purpose interrupt controller solution using the EPIC module of the MPC8240.
- Chapter 13, “Error Handling and Exceptions,” describes how the MPC8240 handles different error conditions.
- Chapter 14, “Power Management,” describes the many hardware support features provided by the MPC8240 for power management.
- Chapter 15, “Debug Features,” describes the MPC8240 features that aid in the process of system bring-up and debug.
- Appendix A, “PowerPC Instruction Set Listings,” lists all the PowerPC instructions while indicating those instructions that are not implemented by the MPC8240; it also includes the instructions that are specific to the MPC8240. Instructions are grouped according to mnemonic, opcode, function, and form. Also included is a quick reference table that contains general information, such as the architecture level, privilege level, and form, and indicates if the instruction is 64-bit and optional.
- Appendix B, “Bit and Byte Ordering,” describes the big- and little-endian modes and provides examples of each.
- This manual also includes a glossary and an index.

Suggested Reading

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

General Information

The following documentation provides useful information about the PowerPC architecture and computer architecture in general:

- The following books are available from the Morgan-Kaufmann Publishers, 340 Pine Street, Sixth Floor, San Francisco, CA 94104; Tel. (800) 745-7323 (U.S.A.), (415) 392-2665 (International); internet address: mkp@mkp.com.
 - *The PowerPC Architecture: A Specification for a New Family of RISC Processors*, Second Edition, by International Business Machines, Inc.
Updates to the architecture specification are accessible via the world-wide web at <http://www.austin.ibm.com/tech/ppc-chg.html>.
 - *PowerPC Microprocessor Common Hardware Reference Platform: A System Architecture*, by Apple Computer, Inc., International Business Machines, Inc., and Motorola, Inc.
 - *Macintosh Technology in the Common Hardware Reference Platform*, by Apple Computer, Inc.
 - *Computer Architecture: A Quantitative Approach*, Second Edition, by John L. Hennessy and David A. Patterson
- *Inside Macintosh: PowerPC System Software*, Addison-Wesley Publishing Company, One Jacob Way, Reading, MA, 01867; Tel. (800) 282-2732 (U.S.A.), (800) 637-0029 (Canada), (716) 871-6555 (International).

PowerPC Documentation

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

- User's manuals—These books provide details about individual PowerPC implementations and are intended to be used in conjunction with The Programming Environments Manual. These include the following:
 - *PowerPC 604™ RISC Microprocessor User's Manual*: MPC604UM/AD (Motorola order #)
 - *MPC603 & EC603e RISC Microprocessor User's Manual*: MPC603UM/AD (Motorola order #)
 - *MPC750 RISC Microprocessor User's Manual*: MPC750UM/AD (Motorola order #)

- Programming environments manuals—This book provides information about resources defined by the PowerPC architecture that are common to PowerPC processors. The 32-bit architecture model is described in:
 - *PowerPC Microprocessor Family: The Programming Environments for 32-Bit Microprocessors*, Rev. 1: MPCFPE32B/AD (Motorola order #)
- *Implementation Variances Relative to Rev. 1 of The Programming Environments Manual* is available via the world-wide web at <http://www.motorola.com/PowerPC/>.
- Addenda/errata to user's manuals—Because some processors have follow-on parts an addendum is provided that describes the additional features and changes to functionality of the follow-on part. These addenda are intended for use with the corresponding user's manuals.
- Hardware specifications—Hardware specifications provide specific data regarding bus timing, signal behavior, and AC, DC, and thermal characteristics, as well as other design considerations for each PowerPC implementation. These include the following:
 - *PowerPC 603e RISC Microprocessor Family: PID7v-603e Hardware Specifications*:
MPC603E7VEC/D (Motorola order #)
 - *PowerPC 603e RISC Microprocessor Family: PID7t-603e Hardware Specifications*:
MPC603E7TEC/D (Motorola order #)
 - *MPC750 RISC Microprocessor Hardware Specifications*
MPC750EC/D (Motorola order #)
 - *MPC8240 Integrated Processor Hardware Specifications*
MPC8240EC/D (Motorola order #)
- Technical Summaries—Each PowerPC implementation has a technical summary that provides an overview of its features. This document is roughly the equivalent to the overview (Chapter 1) of an implementation's user's manual. Technical summaries are available for the 601, 603, 603e, 604, 604e, and EC603e microprocessors which can be ordered as follows:
 - *MPC8240 Integrated Processor Technical Summary*:
MPC8240/D (Motorola order #)
- *PowerPC Microprocessor Family: The Bus Interface for 32-Bit Microprocessors*: MPCBUSIF/AD (Motorola order #) provides a detailed functional description of the 60x bus interface, as implemented on the 601, 603, and 604 family of PowerPC microprocessors. This document is intended to help system and chipset developers by providing a centralized reference source to identify the bus interface presented by the 60x family of PowerPC microprocessors.

- *PowerPC Microprocessor Family: The Programmer's Reference Guide: MPCPRG/D* (Motorola order #) is a concise reference that includes the register summary, memory control model, exception vectors, and the PowerPC instruction set.
- *PowerPC Microprocessor Family: The Programmer's Pocket Reference Guide: MPCPRGREF/D* (Motorola order #)
This foldout card provides an overview of the PowerPC registers, instructions, and exceptions for 32-bit implementations.
- Application notes—These short documents contain useful information about specific design issues useful to programmers and engineers working with PowerPC processors. These include the following:
 - *Instruction and Data Cache Locking on the G2 Processor Core AN1767/D* (Motorola order #)
- Documentation for support chips—These include the following:
 - *MPC105 PCI Bridge/Memory Controller User's Manual: MPC105UM/AD* (Motorola order #)
 - *MPC106 PCI Bridge/Memory Controller User's Manual: MPC106UM/AD* (Motorola order #)

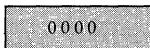
Additional literature on PowerPC implementations is being released as new processors become available. For a current list of PowerPC documentation, refer to the world-wide web at <http://www.mot.com/SPS/PowerPC/>.

Conventions

This document uses the following notational conventions:

mnemonics	Instruction mnemonics are shown in lowercase bold.
<i>italics</i>	Italics indicate variable command parameters, for example, bcctrx . Book titles in text are set in italics.
0x0	Prefix to denote hexadecimal number
0b0	Prefix to denote binary number
rA, rB	Instruction syntax used to identify a source GPR
rA 0	The contents of a specified GPR or the value 0.
rD	Instruction syntax used to identify a destination GPR
frA, frB, frC	Instruction syntax used to identify a source FPR
frD	Instruction syntax used to identify a destination FPR
REG[FIELD]	Abbreviations or acronyms 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 certain contexts, such as a signal encoding, this indicates a don't care.
 - n Used to express an undefined numerical value
 - ¬ NOT logical operator
 - & AND logical operator
 - | OR logical operator
- Indicates reserved bits or bit fields in a register. Although these bits may be written to as either ones or zeros, they are always read as zeros.



Acronyms and Abbreviations

Table i contains acronyms and abbreviations that are used in this document.

Table i. Acronyms and Abbreviated Terms

Term	Meaning
ALU	Arithmetic logic unit
ATE	Automatic test equipment
ASR	Address space register
BAT	Block address translation
BIST	Built-in self test
BIU	Bus interface unit
BPU	Branch processing unit
BUC	Bus unit controller
BUID	Bus unit ID
CAR	Cache address register
CIA	Current instruction address
CMOS	Complementary metal-oxide semiconductor
COP	Common on-chip processor
CR	Condition register
CRTRY	Cache retry queue
CTR	Count register
DAR	Data address register
DBAT	Data BAT
DCMP	Data TLB compare
DEC	Decrementer register
DMISS	Data TLB miss address

Table i. Acronyms and Abbreviated Terms (Continued)

Term	Meaning
DSISR	Register used for determining the source of a DSI exception
DTLB	Data translation lookaside buffer
EA	Effective address
EAR	External access register
ECC	Error checking and correction
FIFO	First-in-first-out
FPR	Floating-point register
FPSCR	Floating-point status and control register
FPU	Floating-point unit
GPR	General-purpose register
HASH1	Primary hash address
HASH2	Secondary hash address
IABR	Instruction address breakpoint register
IBAT	Instruction BAT
ICMP	Instruction TLB compare
IEEE	Institute for Electrical and Electronics Engineers
IMISS	Instruction TLB miss address
IQ	Instruction queue
ITLB	Instruction translation lookaside buffer
IU	Integer unit
L2	Secondary cache
LIFO	Last-in-first-out
LR	Link register
LRU	Least recently used
LSB	Least-significant byte
lsb	Least-significant bit
LSU	Load/store unit
MEI	Modified/exclusive/invalid
MESI	Modified/exclusive/shared/invalid—cache coherency protocol
MMU	Memory management unit
MQ	MQ register
MSB	Most-significant byte

Table i. Acronyms and Abbreviated Terms (Continued)

Term	Meaning
msb	Most-significant bit
MSR	Machine state register
NaN	Not a number
No-op	No operation
OEA	Operating environment architecture
PID	Processor identification tag
PIR	Processor identification register
PLL	Phase-locked loop
POWER	Performance Optimized with Enhanced RISC architecture
PTE	Page table entry
PTEG	Page table entry group
PVR	Processor version register
RAW	Read-after-write
RISC	Reduced instruction set computing
RPA	Required physical address
RTL	Register transfer language
RWITM	Read with intent to modify
SDR1	Register that specifies the page table base address for virtual-to-physical address translation
SLB	Segment lookaside buffer
SPR	Special-purpose register
SR	Segment register
SRR0	Machine status save/restore register 0
SRR1	Machine status save/restore register 1
SRU	System register unit
TAP	Test access port
TB	Time base facility
TBL	Time base lower register
TBU	Time base upper register
TLB	Translation lookaside buffer
TTL	Transistor-to-transistor logic
UIMM	Unsigned immediate value
UISA	User instruction set architecture

Table i. Acronyms and Abbreviated Terms (Continued)

Term	Meaning
UTLB	Unified translation lookaside buffer
UUT	Unit under test
VEA	Virtual environment architecture
WAR	Write-after-read
WAW	Write-after-write
WIMG	Write-through/caching-inhibited/memory-coherency enforced/guarded bits
XATC	Extended address transfer code
XER	Register used for indicating conditions such as carries and overflows for integer operations

Chapter 1 Overview

This chapter provides an overview of the MPC8240 PowerPC™ integrated processor for high-performance embedded systems. The MPC8240 is a cost-effective, general -purpose integrated processor for applications using PCI in networking infrastructure, telecommunications, and other embedded markets. It can be used for control processing in applications such as network routers and switches, mass storage subsystems, network appliances, and print and imaging systems.

For errata or revisions of this document, refer to the website at <http://www.mot.com/powerpc>.

1.1 MPC8240 Integrated Processor Overview

The MPC8240 integrated processor is comprised of a peripheral logic block and a 32-bit superscalar PowerPC processor core, as shown in Figure 1-1.

MPC8240 Integrated Processor Overview

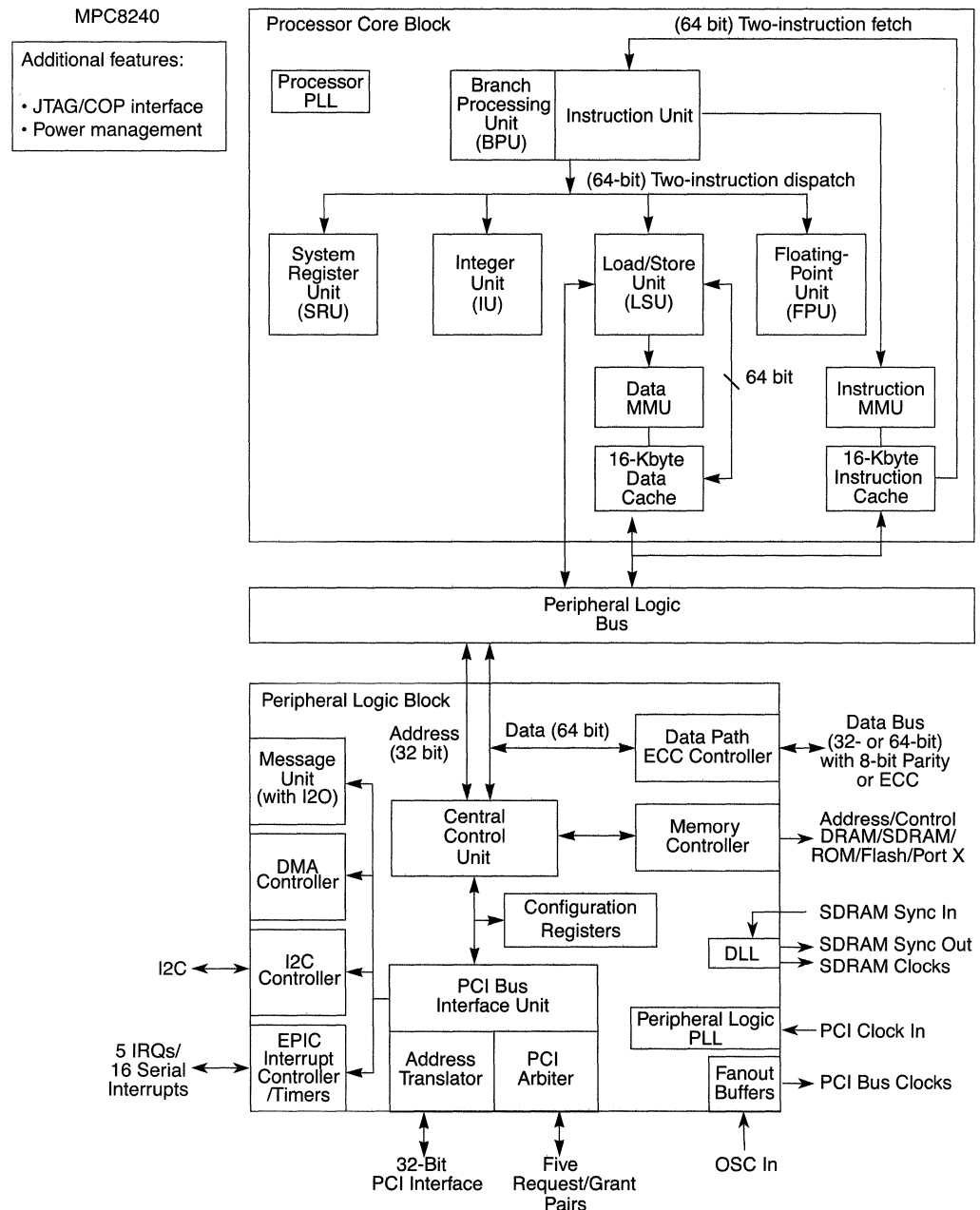


Figure 1-1. MPC8240 Integrated Processor Functional Block Diagram

The peripheral logic integrates a PCI bridge, memory controller, DMA controller, EPIC interrupt controller, a message unit (and I₂O controller), and an I²C controller. The processor core is a full-featured, high-performance processor with floating-point support,

memory management, 16-Kbyte instruction cache, 16-Kbyte data cache, and power management features. The integration reduces the overall packaging requirements and the number of discrete devices required for an embedded system.

The MPC8240 contains an internal peripheral logic bus that interfaces the processor core to the peripheral logic. The 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 peripheral logic PLL. This allows the microprocessor and the peripheral logic block to operate at different frequencies, while maintaining a synchronous bus interface. The interface uses a 64- or 32-bit data bus (depending on memory data bus width) and a 32-bit address bus along with control signals that enable the interface between the processor and peripheral logic to be optimized for performance. PCI accesses to the MPC8240's memory space are passed to the processor bus for snooping when snoop mode is enabled.

The processor core and peripheral logic are general-purpose in order to serve a variety of embedded applications. The MPC8240 can be used as either a PCI host or PCI agent controller.

1.1.1 MPC8240 Integrated Processor Features

Major features of the MPC8240 are as follows:

- Peripheral logic
 - Memory interface
 - Programmable timing supporting either FPM DRAM, EDO DRAM or SDRAM
 - High-bandwidth bus (32/64-bit data bus) to DRAM
 - Supports one to eight banks of 4-, 16-, 64-, or 128-Mbit memory devices
 - Supports 1-Mbyte to 1-Gbyte DRAM memory
 - Contiguous memory mapping
 - 16 Mbytes of ROM space
 - 8-, 32-, or 64-bit ROM
 - Write buffering for PCI and processor accesses
 - Supports normal parity, read-modify-write (RMW), or ECC
 - SDRAM data-path buffer
 - Low voltage TTL logic (LVTTL)
 - Port X: 8-, 32-, or 64-bit general-purpose I/O port using ROM controller interface with address strobe

- 32-bit PCI interface operating up to 66 MHz
 - PCI 2.1-compatible
 - PCI 5.0-V tolerance
 - Support for PCI locked accesses to memory
 - Support for accesses to all PCI address spaces
 - Selectable big- or little-endian operation
 - Store gathering of processor-to-PCI write and PCI-to-memory write accesses
 - Memory prefetching of PCI read accesses
 - Selectable hardware-enforced coherency
 - PCI bus arbitration unit (five request/grant pairs)
- PCI agent mode capability
 - Address translation unit
 - Internal configuration registers accessible from PCI
- Two-channel integrated DMA controller
 - Supports direct mode or chaining mode (automatic linking of DMA transfers)
 - Supports scatter gathering—read or write discontinuous memory
 - Interrupt on completed segment, chain, and error
 - Local-to-local memory
 - PCI-to-PCI memory
 - PCI-to-local memory
 - Local-to-PCI memory
- Message unit
 - Two doorbell registers
 - Inbound and outbound messaging registers
 - I₂O message controller
- I²C controller with full master/slave support
- Embedded programmable interrupt controller (EPIC)
 - Five hardware interrupts (IRQs) or 16 serial interrupts
 - Four programmable timers
- Integrated PCI bus and SDRAM clock generation
- Programmable memory and PCI bus output drivers
- Debug features
 - Memory attribute and PCI attribute signals
 - Debug address signals

- \overline{MIV} signal: Marks valid address and data bus cycles on the memory bus.
- Error injection/capture on data path
- IEEE 1149.1 (JTAG)/test interface
- Processor core
 - High-performance, superscalar processor core
 - Integer unit (IU), floating-point unit (FPU) (user enabled or disabled), load/store unit (LSU), system register unit (SRU), and a branch processing unit (BPU)
 - 16-Kbyte instruction cache
 - 16-Kbyte data cache
 - Lockable L1 cache—entire cache or on a per-way basis
 - Dynamic power management

1.1.2 MPC8240 Integrated Processor Applications

The MPC8240 can be used for control processing in applications such as routers, switches, multi-channel modems, network storage, image display systems, enterprise I/O processor, Internet access device (IAD), disk controller for RAID systems, and copier/printer board control. Figure 1-2 shows the MPC8240 in the role of host processor.

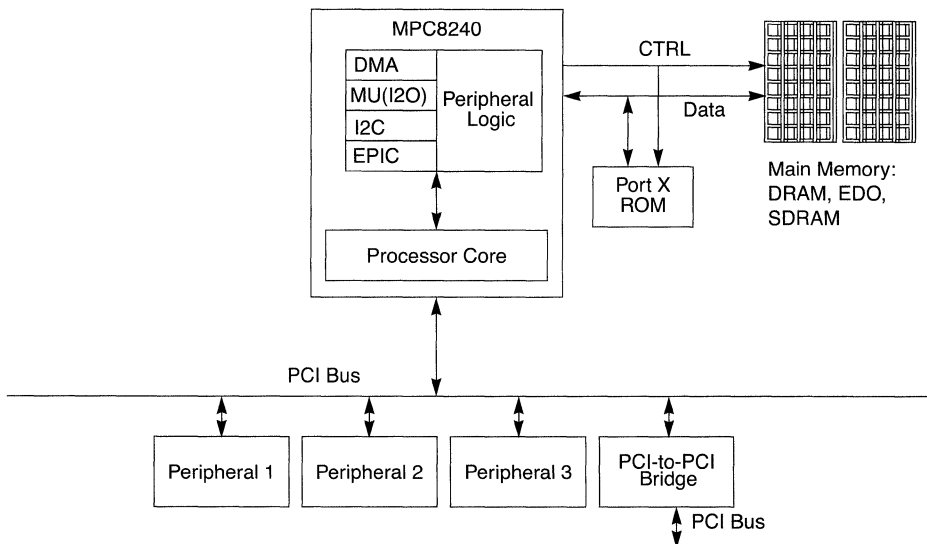
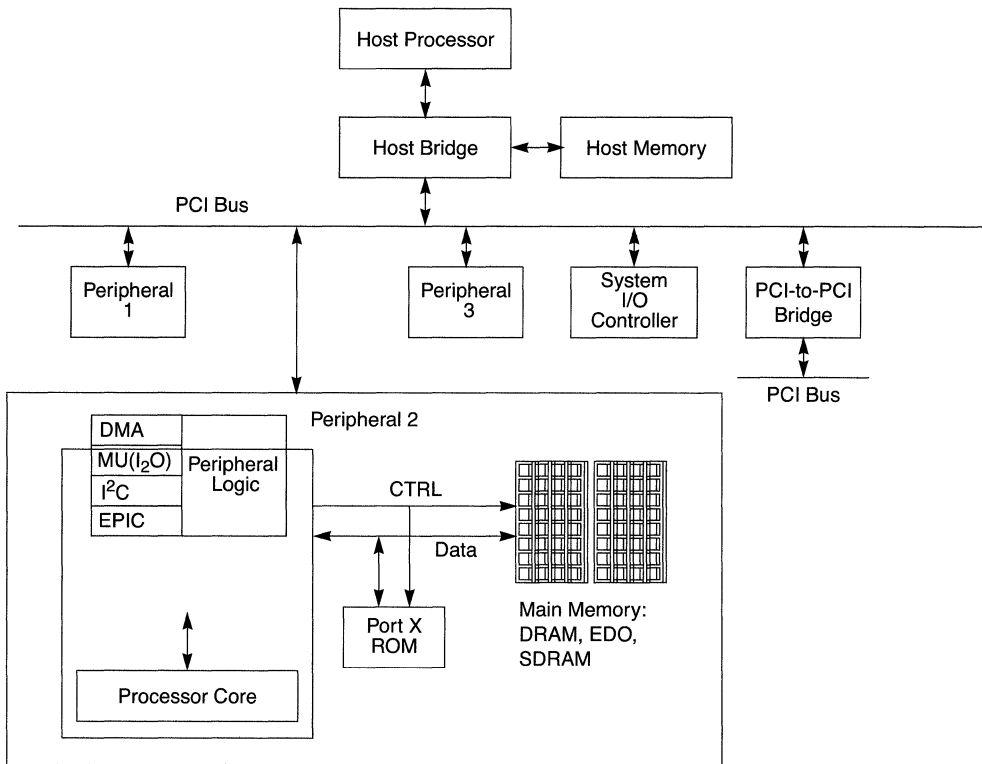


Figure 1-2. System Using an Integrated MPC8240 as a Host Processor

Figure 1-3 shows the MPC8240 in a distributed I/O processor application.



MPC8240

Figure 1-3. Embedded System Using an MPC8240 as a Distributed Processor

Figure 1-4 shows the MPC8240 as a peripheral processing device. The PCI-to-PCI bridge shown could be of the PCI type 0 variety. The MPC8240 would not be part of the system configuration map. This configuration is useful in applications such as RAID controllers, where the I/O devices shown are SCSI controllers, or multi-port network controllers where the devices shown are Ethernet controllers.

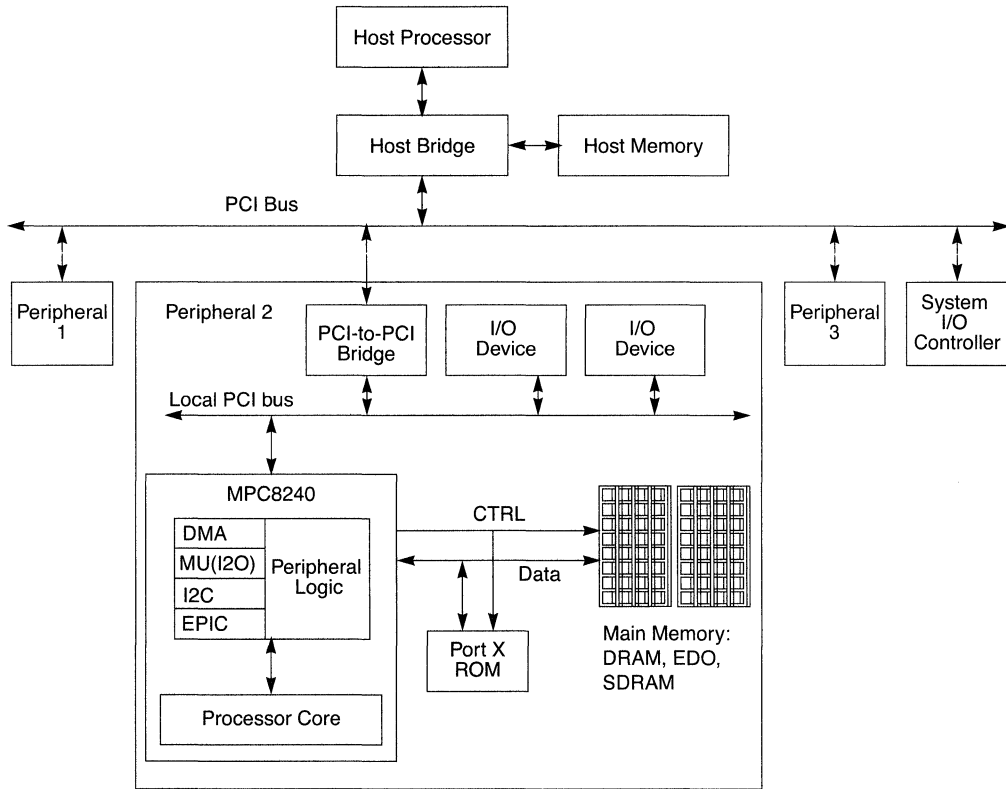


Figure 1-4. Embedded System Using an MPC8240 as a Peripheral Processor

1.2 Processor Core Overview

The MPC8240 contains an embedded version of the MPC603e processor. For detailed information regarding the processor refer to the following:

- *MPC603e & EC603e User's Manual* (Those chapters that describe the programming model, cache model, memory management model, exception model, and instruction timing)
- *PowerPC™ Microprocessor Family: The Programming Environments for 32-Bit Microprocessors*

This section is an overview of the processor core, provides a block diagram showing the major functional units, and describes briefly how those units interact. Chapter 2, "PowerPC Processor Core," provides more important details about the functionality of the MPC8240 processor core.

Processor Core Overview

The processor core is a low-power implementation of the PowerPC microprocessor family of reduced instruction set computing (RISC) microprocessors. The processor core implements the 32-bit portion of the PowerPC architecture, which provides 32-bit effective addresses, integer data types of 8, 16, and 32 bits, and floating-point data types of 32 and 64 bits.

The processor core is a superscalar processor that can issue and retire as many as three instructions per clock. Instructions can execute out of order for increased performance; however, the processor core makes completion appear sequential.

The processor core integrates five execution units—an integer unit (IU), a floating-point unit (FPU), a branch processing unit (BPU), a load/store unit (LSU), and a system register unit (SRU). The ability to execute five instructions in parallel and the use of simple instructions with rapid execution times yield high efficiency and throughput. Most integer instructions execute in one clock cycle. On the processor core, the FPU is pipelined so a single-precision multiply-add instruction can be issued and completed every clock cycle.

The processor core supports integer data types of 8, 16, and 32 bits, and floating-point data types of 32 and 64 bits.

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

As an added feature to the MPC603e core, the MPC8240 can lock the contents of one to three 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 processor core has a selectable 32- or 64-bit data bus and a 32-bit address bus. The processor core supports single-beat and burst data transfers for memory accesses, and supports memory-mapped I/O operations.

Figure 1-5 provides a block diagram of the MPC8240 processor core that shows how the execution units (IU, FPU, BPU, LSU, and SRU) operate independently and in parallel. Note that this is a conceptual diagram and does not attempt to show how these features are physically implemented on the chip.

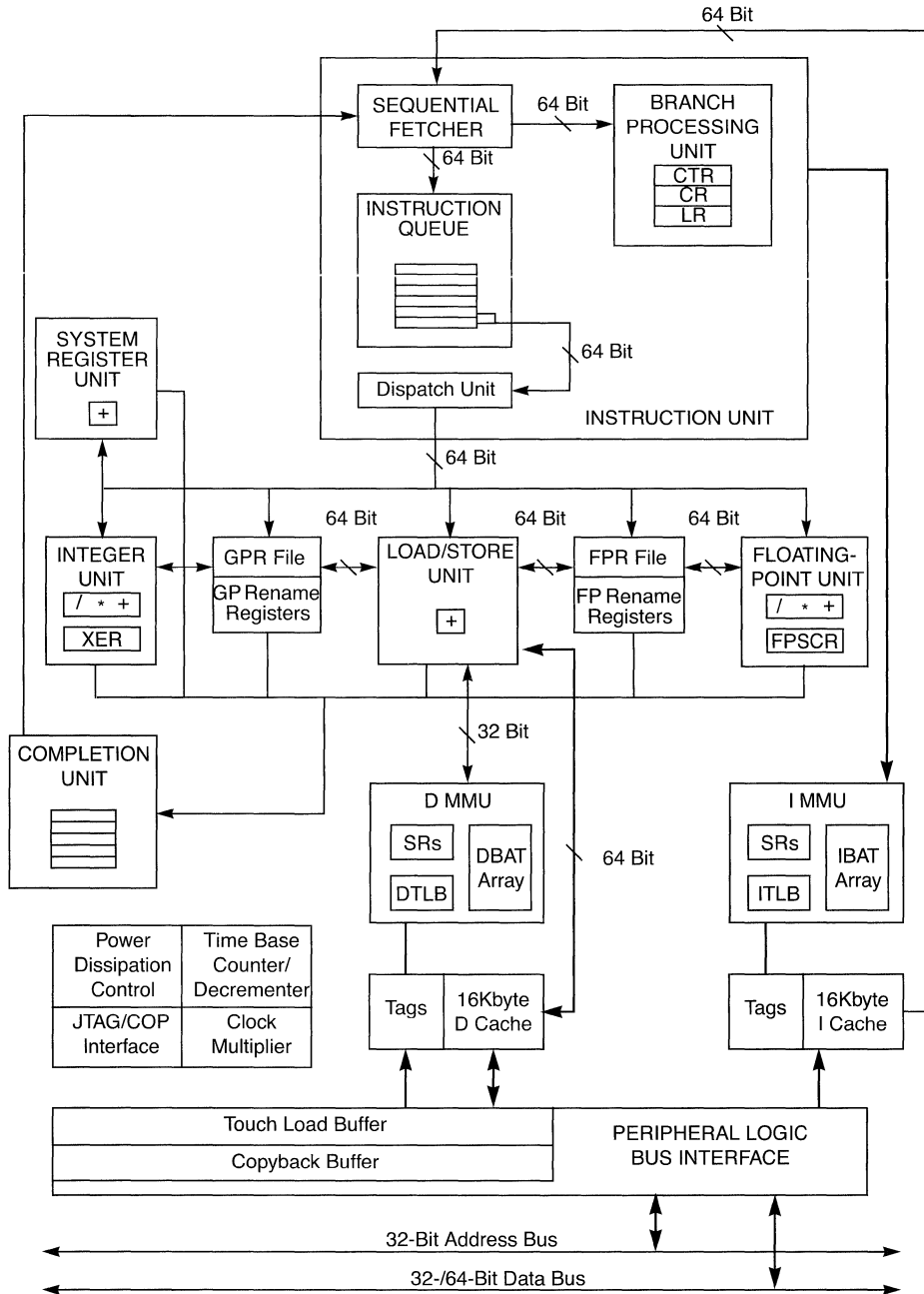


Figure 1-5. MPC8240 Integrated Processor Core Block Diagram

1.3 Peripheral Logic Bus

The MPC8240 contains an internal peripheral logic bus that interfaces the processor core to the peripheral logic. The core can operate at a variety of frequencies allowing the designer to balance performance and power consumption. The processor core is clocked from a separate PLL, which is referenced to the peripheral logic PLL. This allows the microprocessor and the peripheral logic to operate at different frequencies while maintaining a synchronous bus interface.

The processor core-to-peripheral logic interface includes a 32-bit address bus, a 32- or 64-bit data bus as well as control and information signals. The peripheral logic bus allows for internal address-only transactions as well as address and data transactions. The processor core control and information signals include the address arbitration, address start, address transfer, transfer attribute, address termination, data arbitration, data transfer, data termination, and processor state signals. Test and control signals provide diagnostics for selected internal circuits.

The peripheral logic interface supports bus pipelining, which allows the address tenure of one transaction to overlap the data tenure of another. PCI accesses to the memory space are monitored by the peripheral logic bus to allow the processor to snoop these accesses (when snooping not explicitly disabled).

As part of the peripheral logic bus interface, the processor core's data bus is configured at power-up to either a 32- or 64-bit width. When the processor is configured with a 32-bit data bus, memory accesses on the peripheral logic bus interface allow transfer sizes of 8, 16, 24, or 32 bits in one bus clock cycle. Data transfers occur in either single-beat transactions, or two-beat or eight-beat burst transactions, with a single-beat transaction transferring as many as 32 bits. Single- or double-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). Eight-beat burst transactions, which always transfer an entire cache line (32 bytes), are initiated when a line is read from or written to memory.

When the peripheral logic bus interface is configured with a 64-bit data bus, memory accesses allow transfer sizes of 8, 16, 24, 32, 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 line (32 bytes), are initiated when a block is read from or written to memory.

1.4 Peripheral Logic Overview

The peripheral logic block integrates a PCI bridge, memory controller, DMA controller, EPIC interrupt controller/timers, a message unit with an Intelligent Input/Output (I₂O)

message controller, and an Inter-Integrated Circuit (I²C) controller. The integration reduces the overall packaging requirements and the number of discrete devices required for an embedded system.

Figure 1-6 shows the major functional units within the peripheral logic block. Note that this is a conceptual block diagram intended to show the basic features rather than an attempt to show how these features are physically implemented.

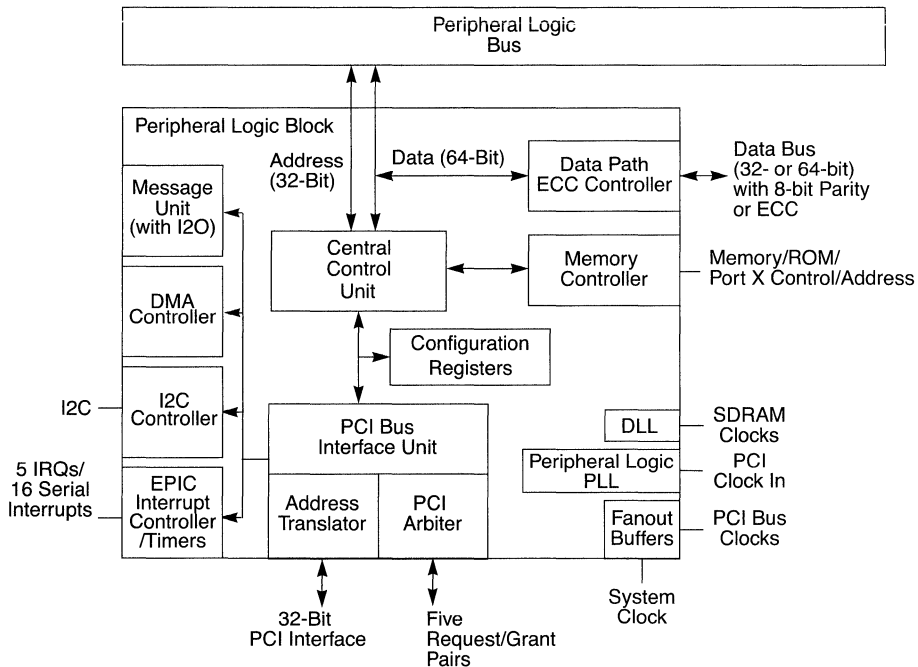


Figure 1-6. MPC8240 Peripheral Logic Block Diagram

1.4.1 Peripheral Logic Features

Major features of the peripheral logic are as follows:

- Peripheral logic bus
 - Supports various operating frequencies and bus divider ratios
 - 32-bit address bus, 64-bit data bus
 - Supports full memory coherency
 - Decoupled address and data buses for pipelining of peripheral logic bus accesses
 - Store gathering on peripheral logic bus-to-PCI writes

Peripheral Logic Overview

- Memory interface
 - 1 Gbyte of RAM space, 16 Mbytes of ROM space
 - High-bandwidth, 64-bit data bus (72 bits including parity or ECC)
 - Supports fast page mode DRAMs, extended data out (EDO) DRAMs, or synchronous DRAMs (SDRAMs)
 - Supports 1 to 8 banks of DRAM/EDO/SDRAM with sizes ranging from 1 to 128 Mbytes per bank
 - Supports page mode SDRAMs—four open pages simultaneously
 - DRAM/EDO configurations support parity or error checking and correction (ECC); SDRAM configurations support ECC
 - ROM space may be split between the PCI bus and the memory bus (8 Mbytes each)
 - Supports 8-bit asynchronous ROM, or 32- or 64-bit burst-mode ROM
 - Supports writing to flash ROM
 - Configurable data path
 - Programmable interface timing
- PCI interface
 - Compatible with PCI Local Bus Specification, Revision 2.1
 - Supports PCI locked accesses to memory using the $\overline{\text{LOCK}}$ signal and protocol
 - Supports accesses to all PCI address spaces
 - Selectable big- or little-endian operation
 - Store gathering on PCI writes to memory
 - Selectable memory prefetching of PCI read accesses
 - Interface operates at up to 66 MHz
 - Parity support
- Supports concurrent transactions on peripheral logic bus and PCI buses

1.4.2 Peripheral Logic Functional Units

The peripheral logic consists of the following major functional units:

- Peripheral logic bus interface
- Memory interface
- PCI interface
 - PCI bus arbitration unit
 - Address maps and translation
 - Big-and little-endian modes

- PCI agent capability
- PCI bus clock buffers and bus ratios
- DMA controller
- Message unit
 - Doorbell registers
 - Message registers
 - I₂O support (circular queues)
- Embedded programmable interrupt controller (EPIC) with four timers
- I²C interface

1.4.3 Memory System Interface

The MPC8240 memory interface controls processor and PCI interactions to main memory. It supports a variety of DRAM, and flash or ROM configurations as main memory. The MPC8240 supports FPM (fast page mode), EDO (extended data out) and synchronous DRAM (SDRAM). The maximum supported memory size is 1 Gbyte of DRAM or SDRAM and 16 Mbytes of ROM/flash. SDRAM must comply with the JEDEC SDRAM specification.

The MPC8240 implements Port X, a flexible memory bus interface that facilitates the connection of general purpose I/O devices. The Port X functionality allows the designer to connect external registers, communication devices, and other such devices directly to the MPC8240. Some devices may require a small amount of external logic to properly generate address strobes, chip selects, and other signals.

The MPC8240 is designed to control a 32-bit or 64-bit data path to main memory DRAM or SDRAM. For a 32-bit data path, the MPC8240 can be configured to check and generate byte parity using four parity bits. For a 64-bit data path, the MPC8240 can be configured to support parity or ECC checking and generation with eight parity/syndrome bits checked and generated.

The MPC8240 supports DRAM or SDRAM bank sizes from 1 to 128 Mbytes and provides bank start address and end address configuration registers, but MPC8240 does not support mixed DRAM/SDRAM configurations. MPC8240 can be configured so that appropriate row and column address multiplexing occurs according to the accessed memory bank. Addresses are provided to DRAM and SDRAM through a 13-bit interface for DRAM and 14-bit interface for SDRAM.

Two bank selects, one write enable, one output enable, and up to 21 address signals are provided for ROM/flash systems.

1.4.4 Peripheral Component Interface (PCI)

The PCI interface for the MPC8240 is compatible with the Peripheral Component Interconnect Specification Revision 2.1. Mode-selectable, big- to little-endian conversion is supplied at the PCI interface. The MPC8240 provides an interface to the PCI bus running at speeds up to 66 MHz.

The MPC8240's PCI interface can be configured as host or agent. In host mode the interface acts as the main memory controller for the system and responds to all host memory transactions.

In agent mode, the MPC8240 can be configured to respond to a programmed window of PCI memory space. A variety of initialization modes are provided to boot the device.

1.4.4.1 PCI Bus Arbitration Unit

The MPC8240 contains a PCI bus arbitration unit, which reduces the need for an equivalent external unit, thus lowering system complexity and cost. It has the following features:

- The unit can be disabled to allow a remote arbitration unit to be used.
- Five external arbitration signal pairs. The MPC8240 is the sixth 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.

1.4.4.2 Address Maps and Translation

The MPC8240's processor bus supports memory-mapped accesses. The address space is divided between memory and PCI according to one of two allowable address maps. An in-bound and out-bound PCI address translation mechanism is provided to support the use of the MPC8240 in agent mode. Note that only map B is supported in agent mode.

When the MPC8240 is used as a peripheral processor and not the primary host, it is possible that only a portion of the memory controlled by it may be visible to the system. In addition, it may be necessary to map the local memory to different system memory address space. The address translation unit handles the mapping of both in-bound and out-bound transactions for these cases.

1.4.4.3 Byte Ordering

The MPC8240 allows the processor to run in either big- or little-endian mode (except for the initial boot code which must run in big-endian mode).

1.4.4.4 PCI Agent Capability

In certain applications the embedded system architecture dictates that the MPC8240 act as peripheral processor. In this case the peripheral logic must not act like a host bridge for the PCI bus. Instead it functions as a configurable device that is accessed by a host bridge. This capability allows multiple MPC8240 devices to coexist with other PCI peripheral devices on a single PCI bus. The MPC8240 contains PCI 2.1-compatible configuration capabilities.

1.4.5 DMA Controller

The integrated DMA controller contains two independent units, each capable of performing the following types of transfers:

- PCI-to-local memory
- Local-to-PCI memory
- PCI-to-PCI memory
- Local-to-local memory

The DMA controller allows chaining through local memory-mapped chain descriptors. Transfers can be scatter gathered and misaligned. Interrupts are provided on completed segment, chain, and error conditions.

1.4.6 Message Unit (MU)

Many embedded applications require handshake algorithms to pass control, status, and data information from one owner to another. This is made easier with doorbell and message registers. The MPC8240 has a message unit (MU) that implements doorbell and message registers as well as an I₂O interface. The MU has many conditions that can cause interrupts and uses the EPIC unit to signal external interrupts to the PCI interface and internal interrupts to the processor core.

1.4.6.1 Doorbell Registers

The MPC8240 MU contains one 32-bit inbound doorbell register and one 32-bit outbound doorbell register. The inbound doorbell register allows a remote processor to set a bit in the register from the PCI bus. This, in turn, generates an interrupt to the processor core.

The processor core can write to the outbound register, causing the outbound interrupt signal $\overline{\text{INTA}}$ to assert, thus interrupting the host processor. Once $\overline{\text{INTA}}$ is generated, it can be cleared only by the host processor by writing ones to the bits that are set in the outbound doorbell register.

1.4.6.2 Inbound and Outbound Message Registers

The MPC8240 contains two 32-bit inbound message registers and two 32-bit outbound message registers. The inbound registers allow a remote host or PCI master to write a 32-bit value, causing an interrupt to the processor core. The outbound registers allow the processor core to write an outbound message which causes the outbound interrupt signal $\overline{\text{INTA}}$ to assert.

1.4.6.3 Intelligent Input/Output Controller (I₂O)

The intelligent I/O specification is an open standard that defines an abstraction layer interface between the OS and subsystem drivers. Messages are passed between the message abstraction layer from one device to another.

The I₂O specification describes a system as being made up of host processors and input/output platforms (IOPs). The host processor is a single processor or a collection of processors working together to execute a homogenous operating system. An IOP consists of a processor, memory, and I/O interfaces. The IOP functions separately from other processors within the system to handle system I/O functions.

The I₂O controller of the MU enhances communication between hosts and IOPs within a system. The MU maintains a set of FIFO buffers located in local IOP memory. Four queues are provided: two for inbound messages and two for outbound messages. The inbound message queues are used to transfer messages from a remote host or IOP to the processor core. The outbound queues are used to transfer messages from the processor core to the remote host. Messages are transferred between the host and the IOP using PCI memory-mapped registers. The MPC8240's I₂O controller facilitates moving the messages to and from the inbound and outbound registers and local IOP memory. Interrupts signal the host and IOP to indicate the arrival of new messages.

1.4.7 Inter-Integrated Circuit (I²C) Controller

The I²C serial interface has become an industry de-facto standard for communicating with low-speed peripherals. Typically, it is used for system management functions and EEPROM support. The MPC8240 contains an I²C controller with full master and slave functionality.

1.4.8 Embedded Programmable Interrupt Controller (EPIC)

The integrated hardware interrupt controller reduces the overall component count in embedded applications. The embedded programmable interrupt controller (EPIC) is designed to collect external and internal hardware interrupts, prioritize them, and deliver them to the processor core.

The module operates in two modes:

- In direct mode, five level- or edge-triggered interrupts can be connected directly to an MPC8240.
- The MPC8240 provides a serial delivery mechanism for when more than five external interrupt sources are needed. The serial mechanism allows for up to 16 interrupts to be serially scanned into the MPC8240. This mechanism increases the number of interrupts without increasing the number of pins.

The outbound interrupt request signal, $\overline{L_INT}$, is used to signal interrupts to the host processor when the MPC8240 is configured for agent mode. The MPC8240 EPIC includes four programmable timers that can be used for system timing or for generating periodic interrupts.

1.4.9 Integrated PCI Bus and SDRAM Clock Generation

There are two clocking solutions directed towards different system requirements. For systems where the MPC8240 is the host controller with a minimum number of clock loads, clock fan-out buffers are provided on chip.

For systems requiring more clock fan out or where the MPC8240 is an agent device, external clock buffers may be used.

1.4.10 Bus Ratios

The MPC8240 requires a single clock input signal, `PCI_SYNC_IN`, which can be driven by the PCI clock fan-out buffers—specifically the `PCI_SYNC_OUT` output. `PCI_SYNC_IN` can also be driven by an external clock driver.

`PCI_SYNC_IN` is driven by the PCI bus frequency. An internal PLL, using `PCI_SYNC_IN` as a reference, generates an internal peripheral bus clock that is used for the internal logic. The peripheral bus clock frequency is selectable by the MPC8240 PLL configuration signals (`PLL_CFG[0–4]`) to be a multiple of the `PCI_SYNC_IN` frequency.

The MPC8240 provides an on-chip delay-locked loop (DLL) that can supply an external memory bus clock to SDRAM banks. The memory bus clock is of the same frequency and synchronous with the internal peripheral bus clock.

The internal clocking of the processor core is generated from and synchronized to the internal peripheral bus clock by means of a second PLL. The core's PLL provides multiples of the internal processor core clock rates as specified in the *MPC8240 Hardware Specification*.

1.5 Power Management

The MPC8240 provides both automatic and program-controllable power reduction modes for progressive reduction of power consumption.

The MPC8240 has independent power management functionality for both the processor core and the peripheral logic. The MPC8240 provides hardware support for three levels of programmable power reduction for both the processor and the peripheral logic. Doze, nap, and sleep modes are invoked by register programming—`HID0` in the case of the processor core and configuration registers in the case of the peripheral logic block. The processor and peripheral logic blocks are both fully static, allowing internal logic states to be preserved during all power-saving modes. The following sections describe the programmable power modes.

1.5.1 Programmable Processor Power Management Modes

Table 1-1 summarizes the programmable power-saving modes for the processor core. These are very similar to those available in the MPC603e device.

Table 1-1. Programmable Processor Power Modes

PM Mode	Functioning Units	Activation Method	Full-Power Wake Up Method
Full power	All units active	—	—
Full power (with DPM)	Requested logic by demand	By instruction dispatch	—
Doze	Bus snooping Data cache as needed Decrementer timer	Controlled by software (write to HID0)	External asynchronous exceptions (assertion of \overline{SMI} or \overline{int}), Decrementer exception Hard or soft reset Machine check exception (\overline{mcp})
Nap	Decrementer timer	Controlled by software (write to HID0) and qualified with \overline{QACK} from peripheral logic	External asynchronous exceptions (assertion of \overline{SMI} , or \overline{int}) Decrementer exception Negation of \overline{QACK} by peripheral logic Hard or soft reset Machine check exception (\overline{mcp})
Sleep	None	Controlled by software (write to HID0) and qualified with \overline{QACK} from peripheral logic	External asynchronous exceptions (assertion of \overline{SMI} , or \overline{int}) Negation of \overline{QACK} by peripheral logic Hard or soft reset Machine check exception (\overline{mcp})

1.5.2 Peripheral Logic Power Management Modes

The following subsections describe the power management modes of the peripheral logic. Table 1-2 summarizes the programmable power-saving modes for the peripheral logic block.

Table 1-2. Programmable Peripheral Logic Power Modes

PM Mode	Functioning Units	Activation Method	Full-Power Wake Up Method
Full power	All units active	—	—
Doze	PCI address decoding and bus arbiter System RAM refreshing Processor bus request and NMI monitoring EPIC unit I ² C unit PLL	Controlled by software (write to PMCR)	PCI access to memory Processor bus request Assertion of NMI Interrupt to EPIC Hard Reset

Table 1-2. Programmable Peripheral Logic Power Modes (Continued)

PM Mode	Functioning Units	Activation Method	Full-Power Wake Up Method
Nap	PCI address decoding and bus arbiter System RAM refreshing Processor bus request and NMI monitoring EPIC unit I ² C uni PLL	Controlled by software (write to PMCR) and processor core in nap or sleep mode	PCI access to memory ¹ Processor bus request Assertion of NMI Interrupt to EPIC Hard Reset
Sleep	PCI bus arbiter System RAM refreshing (can be disabled) Processor bus request and NMI monitoring EPIC unit I ² C unit PLL (can be disabled)	Controlled by software (write to PMCR) and processor core in nap or sleep mode	Processor bus request Assertion of NMI Interrupt to EPIC Hard Reset

¹A PCI access to memory in nap mode does not cause \overline{QACK} to negate; subsequently, it does not wake up the processor core. Thus the processor won't snoop this access. After servicing the PCI access, the peripheral logic automatically returns to the nap mode.

1.6 Debug Features

The MPC8240 includes the following debug features:

- Memory attribute and PCI attribute signals
- Debug address signals
- \overline{MIV} signal: Marks valid address and data bus cycles on the memory bus.
- Error injection/capture on data path
- IEEE 1149.1 (JTAG)/test interface

1.6.1 Memory Attribute and PCI Attribute Signals

The MPC8240 provides additional information corresponding to memory and PCI activity on several signals to assist with system debugging. The two types of attribute signals are described as follows:

- The memory attribute signals are associated with the memory interface and provide information concerning the source of the memory operation being performed by the MPC8240.
- The PCI attribute signals are associated with the PCI interface and provide information concerning the source of the PCI operation being performed by the MPC8240.

1.6.2 Memory Debug Address

When enabled, the debug address provides software disassemblers a simple way to reconstruct the 30-bit physical address for a memory bus transaction to DRAM and

Debug Features

SDRAM, ROM, FLASH, or PortX. For DRAM or SDRAM, these 16 debug address signals are sampled with the column address and chip-selects. For ROMs, FLASH, and PortX devices, the debug address pins are sampled at the same time as the ROM address and can be used to recreate the 24-bit physical address in conjunction with ROM address. The granularity of the reconstructed physical address is limited by the bus width of the interface; double-words for 64-bit interfaces, words for 32-bit interfaces, and bytes for 8-bit interfaces.

1.6.3 Memory Interface Valid (\overline{MIV})

The memory interface valid signal, \overline{MIV} , is asserted whenever FPM, EDO, SDRAM, FLASH, or ROM addresses or data are present on the external memory bus. It is intended to help reduce the number of bus cycles that logic analyzers must store in memory during a debug trace.

1.6.4 Error Injection/Capture on Data Path

The MPC8240 provides hardware to exercise and debug the ECC and parity logic by allowing the user to inject multi-bit stuck-at faults onto the peripheral logic or memory data/parity busses and to capture the data/parity output on receipt of an ECC or parity error.

1.6.5 IEEE 1149.1 (JTAG)/Test Interface

The processor core provides IEEE 1149.1 functions for facilitating board testing and debugging. The IEEE 1149.1 test interface provides a means for boundary-scan testing the processor core and the board to which it is attached.

Chapter 2

PowerPC Processor Core

The MPC8240 contains an embedded version of the PowerPC 603e™ processor called the G2 core. This chapter provides an overview of the basic functionality of the G2 processor core. For detailed information regarding the processor refer to the following:

- *MPC603e & EC603e User's Manual* (Those chapters that describe the programming model, cache model, memory management model, exception model, and instruction timing)
- *PowerPC™ Microprocessor Family: The Programming Environments for 32-Bit Microprocessors*

This section describes the details of the processor core, provides a block diagram showing the major functional units, and describes briefly how those units interact. At the end of this chapter, there is a section that outlines the detailed differences between the G2 core and the MPC8240 processor.

The signals associated with the processor core are described in Chapter 3, “Signal Descriptions and Clocking.”

2.1 Overview

The processor core is a low-power implementation of the PowerPC microprocessor family of reduced instruction set computing (RISC) microprocessors. The processor core implements the 32-bit portion of the PowerPC architecture, which supports 32-bit effective addresses.

Figure 2-1 is a block diagram of the processor core.

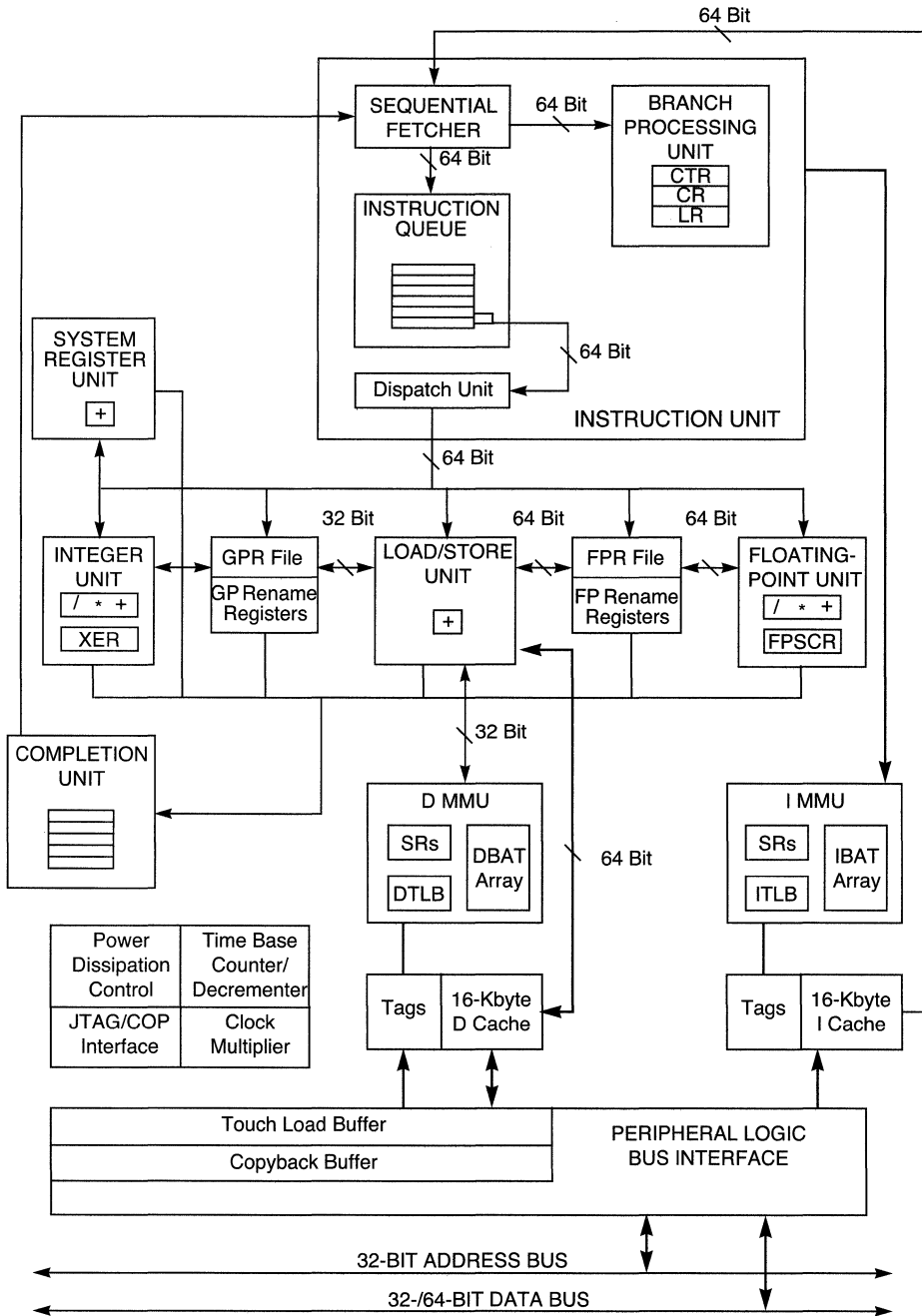


Figure 2-1. MPC8240 Integrated Processor Core Block Diagram

The processor core is a superscalar processor that can issue and retire as many as three instructions per clock. Instructions can execute out of order for increased performance; however, the processor core makes completion appear sequential.

The processor core integrates five execution units—an integer unit (IU), a floating-point unit (FPU), a branch processing unit (BPU), a load/store unit (LSU), and a system register unit (SRU). The ability to execute five instructions in parallel and the use of simple instructions with rapid execution times yield high efficiency and throughput. Most integer instructions execute in one clock cycle. On the processor core, the FPU is pipelined so a single-precision multiply-add instruction can be issued and completed every clock cycle.

The processor core supports integer data types of 8, 16, and 32 bits, and floating-point data types of 32 and 64 bits.

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

As an added feature to the MPC603e core, the MPC8240 can lock the contents of 1–3 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 processor core has a selectable 32- or 64-bit data bus and a 32-bit address bus. The processor core supports single-beat and burst data transfers for memory accesses and supports memory-mapped I/O operations.

2.2 PowerPC Processor Core Features

This section describes the major features of the processor core:

- High-performance, superscalar microprocessor
 - As many as three instructions issued and retired per clock cycle
 - As many as five instructions in execution per clock cycle
 - Single-cycle execution for most instructions
 - f FPU for all single-precision and most double-precision operations

PowerPC Processor Core Features

- Five independent execution units and two register files
 - BPU featuring static branch prediction
 - A 32-bit IU
 - Fully IEEE 754-compliant FPU for both single- and double-precision operations
 - LSU for data transfer between data cache and GPRs and FPRs
 - SRU that executes condition register (CR), special-purpose register (SPR), and integer add/compare instructions
 - Thirty-two GPRs for integer operands
 - Thirty-two 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
 - BPU that performs CR lookahead operations
 - Instruction fetch unit capable of fetching two instructions per clock from the instruction cache
 - A six-entry instruction queue that provides lookahead capability
 - Independent pipelines with feed-forwarding that reduces data dependencies in hardware
 - 16-Kbyte data cache—four-way set-associative, physically addressed, LRU replacement algorithm
 - 16-Kbyte instruction cache—four-way set-associative, physically addressed, LRU replacement algorithm
 - Cache write-back or write-through operation programmable on a per page or per block basis
 - Address translation facilities for 4-Kbyte page size, variable block size, and 256-Mbyte segment size
 - A 64-entry, two-way set-associative ITLB
 - A 64-entry, two-way set-associative DTLB
 - Four-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 32- or 64-bit split-transaction internal data bus interface to the peripheral logic bus with bursting
 - Support for one-level address pipelining and out-of-order bus transactions

- Hardware support for misaligned little-endian accesses
- Configurable processor bus frequency multipliers as defined in the *MPC8240 Integrated Processor Hardware Specifications*.
- Integrated power management
 - Three power-saving modes: doze, nap, and sleep
 - Automatic dynamic power reduction when internal functional units are idle
- Deterministic behavior and debug features
 - On-chip cache locking options for the instruction and data caches (1–3 ways or the entire cache contents can be locked)
 - In-system testability and debugging features through JTAG and boundary-scan capability

Figure 2-1 shows how the execution units (IU, BPU, FPU, 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.

2.2.1 Instruction Unit

As shown in Figure 2-1, the instruction unit, which contains a fetch unit, instruction queue, dispatch unit, and the 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 extracts branch instructions from the fetcher and uses static branch prediction on unresolved conditional branches to allow the instruction unit to fetch instructions from a predicted target instruction stream while a conditional branch is evaluated. The BPU folds out branch instructions for unconditional branches or conditional branches unaffected by instructions in progress in the execution pipeline.

Instructions issued beyond a predicted branch do not complete execution until the branch is resolved, preserving the programming model of sequential execution. If any of these instructions are to be executed in the BPU, they are decoded but not issued. Instructions to be executed by the IU, FPU, LSU, and SRU are issued and allowed to complete up to the register write-back stage. Write-back is allowed when a correctly predicted branch is resolved, and instruction execution continues without interruption on the predicted path. If branch prediction is incorrect, the instruction unit flushes all predicted path instructions, and instructions are issued from the correct path.

2.2.2 Instruction Queue and Dispatch Unit

The instruction queue (IQ), shown in Figure 2-1, 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 the 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. Reservation stations at the IU, FPU, LSU, and SRU facilitate instruction dispatch to those units. The dispatch unit checks for source and destination register dependencies, determines dispatch serializations, and inhibits subsequent instruction dispatching as required. Section 2.7, “Instruction Timing,” describes instruction dispatch in detail.

2.2.3 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, instructions are fetched 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 condition register (CR). The BPU calculates the return pointer for subroutine 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 other instructions.

2.2.4 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.

In addition to the BPU, the processor core provides four other execution units and a completion unit, which are described in the following sections.

2.2.4.1 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. Thirty-two general-purpose registers are provided to support integer operations. Stalls due to contention for GPRs are minimized by the automatic allocation of rename registers. The processor core writes the contents of the rename registers to the appropriate GPR when integer instructions are retired by the completion unit.

2.2.4.2 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 processor to efficiently implement multiply and multiply-add operations. The FPU is pipelined so that single-precision instructions and double-precision instructions can be issued back-to-back. Thirty-two floating-point registers are provided to support floating-point operations. Stalls due to contention for FPRs are minimized by the automatic allocation of rename registers. The processor writes the contents of the rename registers to the appropriate FPR when floating-point instructions are retired by the completion unit.

The processor supports all IEEE 754 floating-point data types (normalized, denormalized, NaN, zero, and infinity) in hardware, eliminating the latency incurred by software exception routines.

2.2.4.3 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 translated in program order; however, the actual memory accesses can occur out of order. Synchronizing instructions are provided to enforce strict ordering where needed.

Cacheable loads, when free of data dependencies, execute in an out-of-order manner with a maximum throughput of one per cycle and 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. Store operations do not occur until a predicted branch is resolved. They remain in the store queue until the completion logic signals that the store operation is definitely to be completed to memory.

The processor core executes store instructions with a maximum throughput of one per cycle and a three-cycle total latency. The time required to perform the actual load or store operation varies depending on whether the operation involves the cache, system memory, or an I/O device.

2.2.4.4 System Register Unit (SRU)

The SRU executes various system-level instructions, including condition register logical operations and move to/from special-purpose register instructions, and integer add/compare instructions. Because SRU instructions affect modes of processor operation, most SRU instructions 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 the instruction completes.

2.2.5 Completion Unit

The completion unit tracks instructions from dispatch through execution, and then retires, or completes them in program order. Completing an instruction commits the processor core to any architectural register changes caused by that instruction. In-order completion ensures the correct architectural state when the processor core must recover from a mispredicted branch or any exception.

Instruction state and other information required for completion is kept in a first-in-first-out (FIFO) queue of five completion buffers. A single completion buffer is allocated for each instruction once it enters the dispatch unit. An available completion buffer is a required resource for instruction dispatch; if no completion buffers are available, instruction dispatch stalls. A maximum of two instructions per cycle are completed in order from the queue.

2.2.6 Memory Subsystem Support

The processor core supports cache and memory management through separate instruction and data MMUs (IMMU and DMMU). The processor core also provides dual 16-Kbyte instruction and data caches, and an efficient peripheral bus interface to facilitate access to main memory and other bus subsystems. The memory subsystem support functions are described in the following subsections.

2.2.6.1 Memory Management Units (MMUs)

The processor core's MMUs support up to 4 Petabytes (2^{52}) of virtual memory and 4 Gbytes (2^{32}) of physical memory (referred to as real memory in the PowerPC architecture specification) for instructions 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. 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 the effective addresses for instruction fetching.

The MMUs translate effective addresses and enforce the protection hierarchy programmed by the operating system in relation to the supervisor/user privilege level of the access and in relation to the type of access—load or store.

2.2.6.2 Cache Units

The processor core provides independent 16-Kbyte, four-way set-associative instruction and data caches. The cache block size is 32 bytes. The caches are designed to adhere to a write-back policy, but the processor core allows control of cacheability, write policy, and memory coherency at the page and block levels. The caches use a least recently used (LRU) replacement algorithm.

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.

Note that the MPC8240 processor core has some additional cache locking functionality compared to the MPC603e. This is described in more detail in Section 2.4.2.3, “Cache Locking.”

2.2.6.3 Peripheral Logic Bus Interface

The MPC8240 contains an internal peripheral logic bus that interfaces the processor core to the peripheral logic. This internal bus is very similar in function to the external 60x bus interface on the MPC603e. In the case of the MPC8240, the central control unit (CCU) terminates all the transactions and internally directs all accesses to the appropriate peripheral (or memory) interface.

2.2.6.3.1 Peripheral Logic Bus Protocol

The processor core-to-peripheral logic interface includes a 32-bit address bus, a 32- or 64-bit data bus as well as control and information signals. The peripheral logic interface allows for address-only transactions as well as address and data transactions. The processor core control and information signals include the address arbitration, address start, address transfer, transfer attribute, address termination, data arbitration, data transfer, data termination, and processor state signals. Test and control signals provide diagnostics for selected internal circuits.

The peripheral logic interface supports bus pipelining, which allows the address tenure of one transaction to overlap the data tenure of another. PCI accesses to the memory space are monitored by the peripheral logic bus to allow the processor to snoop these accesses (provided PICR[27] is cleared).

2.2.6.3.2 Peripheral Logic Bus Data Transfers

As part of the peripheral logic bus interface, the processor core’s data bus is configured at power-up (by the value on the DL[0] signal) to either a 32- or 64-bit width.

When the processor is configured with a 32-bit data bus, memory accesses on the peripheral logic bus interface allow transfer sizes of 8, 16, 24, or 32 bits in one bus clock cycle. Data transfers occur in either single-beat transactions, or two-beat or eight-beat burst transactions, with a single-beat transaction transferring as many as 32 bits. Single- or double-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). Eight-beat burst transactions, which always transfer an entire cache line (32 bytes), are initiated when a line is read from or written to memory.

When the peripheral logic bus interface is configured with a 64-bit data bus, memory accesses allow transfer sizes of 8, 16, 24, 32, 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 line (32 bytes), are initiated when a line is read from or written to memory.

2.2.6.3.3 Peripheral Logic Bus Frequency

The 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 peripheral logic PLL. This allows the microprocessor and the peripheral logic to operate at different frequencies while maintaining a synchronous bus interface.

2.3 Programming Model

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

2.3.1 Register Set

This section describes the register organization in the processor core as defined by the three programming environments of the PowerPC architecture—the user instruction set architecture (UISA), the virtual environment architecture (VEA), and the operating environment architecture (OEA), as well as the MPC8240 core implementation-specific registers. Full descriptions of the basic register set defined by the PowerPC architecture are provided in Chapter 2, “PowerPC Register Set,” in The Programming Environments Manual.

The PowerPC architecture defines register-to-register operations for all computational instructions. Source data for these instructions is accessed from the on-chip registers or is provided as an immediate value embedded in the opcode. The three-register instruction format allows specification of a target register distinct from the two source registers, thus preserving the original data for use by other instructions and reducing the number of instructions required for certain operations. Data is transferred between memory and registers with explicit load and store instructions only.

Figure 2-2 shows the complete MPC8240 register set and the programming environment to which each register belongs. This figure includes both the PowerPC register set and the MPC8240-specific registers.

Note that there may be registers common to other PowerPC processors that are not implemented in the MPC8240's processor core. Unsupported Special Purpose Register (SPR) values are treated as follows:

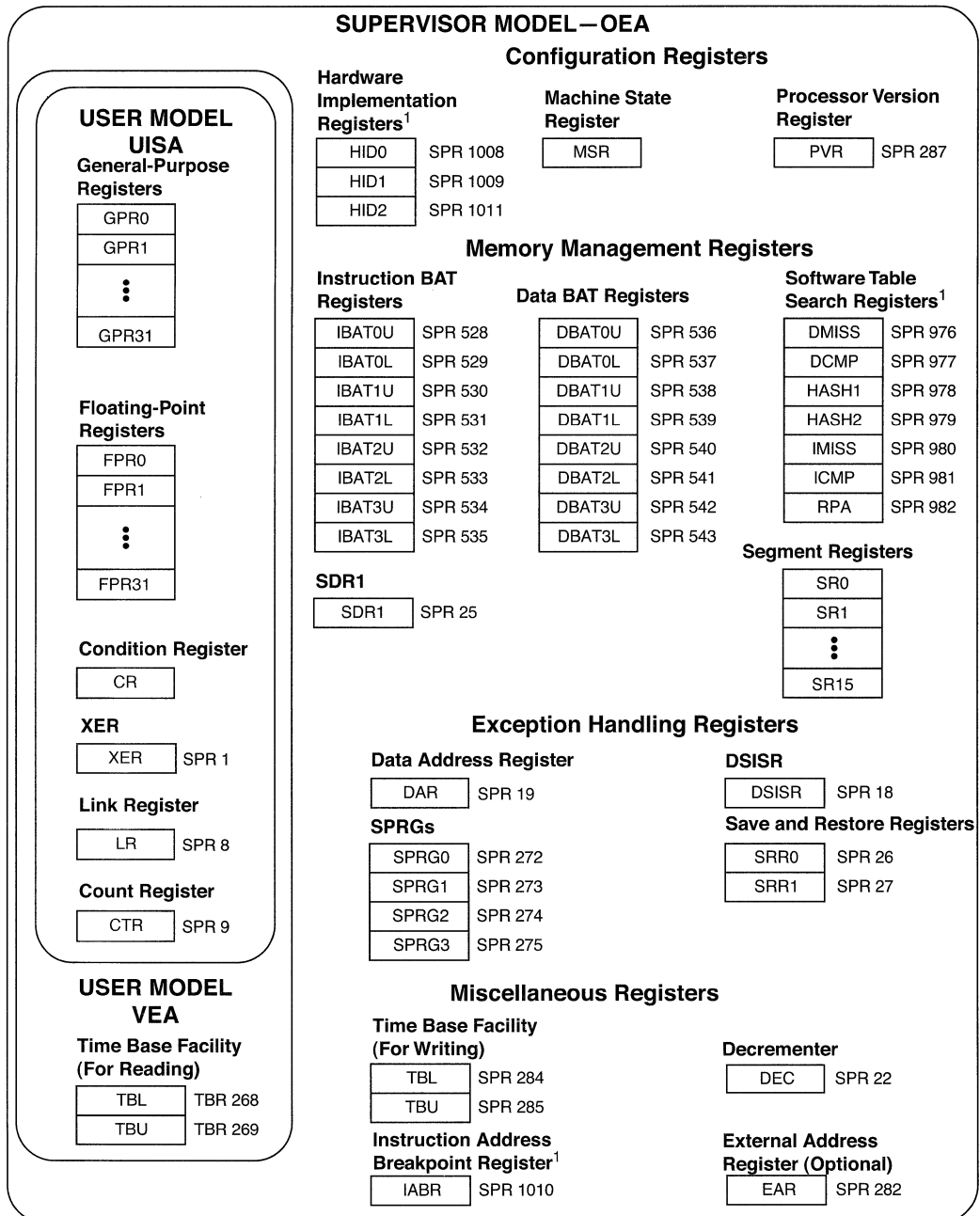
- Any **mtspr** with an invalid SPR executes as a no-op.
- Any **mfspr** with an invalid SPR causes boundedly undefined results in the target register.

Conversely, some SPRs in the processor core may not be implemented at all or may not be implemented in the same way in other PowerPC processors.

2.3.1.1 PowerPC Register Set

The PowerPC UISA registers, shown in Figure 2-2, can be accessed by either user- or supervisor-level instructions. The general-purpose registers (GPRs) and floating-point registers (FPRs) are accessed through instruction operands. Access to registers can be explicit (that is, through the use of specific instructions for that purpose such as the **mtspr** and **mfspr** instructions) or implicit as part of the execution (or side effect) of an instruction. Some registers are accessed both explicitly and implicitly.

The number to the right of the register name indicates the number that is used in the syntax of the instruction operands to access the register (for example, the number used to access the XER is one). For more information on the PowerPC register set, refer to Chapter 2, "PowerPC Register Set," in The Programming Environments Manual.



USER MODEL

UISA

General-Purpose Registers

GPR0
GPR1
⋮
GPR31

Floating-Point Registers

FPR0
FPR1
⋮
FPR31

Condition Register

CR

XER

XER	SPR 1
-----	-------

Link Register

LR	SPR 8
----	-------

Count Register

CTR	SPR 9
-----	-------

USER MODEL

VEA

Time Base Facility (For Reading)

TBL	TBR 268
TBU	TBR 269

¹ These implementation-specific registers may not be supported by other PowerPC processors or processor cores.

Figure 2-2. MPC8240 Programming Model—Registers

2.3.1.2 MPC8240-Specific Registers

The set of registers specific to the MPC603e are shown in Figure 2-2. Most of these are described in the MPC603e User’s Manual and are implemented in the MPC8240 as follows:

- MMU software table search registers—DMISS, DCMP, HASH1, HASH2, IMISS, ICMP, and RPA. These registers facilitate the software required to search the page tables in memory.
- IABR—This register facilitates the setting of instruction breakpoints.

The hardware implementation-dependent registers (HIDx) are implemented differently in the MPC8240 as described in the following subsections.

2.3.1.2.1 Hardware Implementation-Dependent Register 0 (HID0)

The processor core’s implementation of HID0 differs from the MPC603e User’s Manual as follows:

- Bit 5, HID0[EICE], has been removed
- No support for pipeline tracking

Figure 2-3 shows the MPC8240 implementation of HID0. HID0 can be accessed with `mtspr` and `mfspr` using SPR1008.

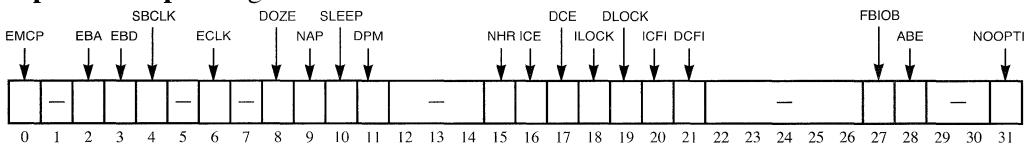


Figure 2-3. Hardware Implementation Register 0 (HID0)

Table 2-1 shows the bit definitions for HID0.

Table 2-1. HID0 Field Descriptions

Bits	Name	Description
0	EMCP	Enable machine check internal signal 0 The assertion of the internal <code>mcp</code> signal from the peripheral logic does not cause a machine check exception. 1 Enables the machine check exception based on assertion of the internal <code>mcp</code> signal from the peripheral logic to the processor core. Note that the machine check exception is further affected by <code>MSR[ME]</code> , which specifies whether the processor checkstops or continues processing.
1	—	Reserved

Table 2-1. HID0 Field Descriptions (Continued)

Bits	Name	Description
2	EBA	Enable/disable internal peripheral bus (60x bus) address parity checking 0 Prevents address parity checking 1 Allows a address parity error to cause a checkstop if MSR[ME] = 0 or a machine check exception if MSR[ME] = 1 EBA and EBD let the processor operate with memory subsystems that do not generate parity.
3	EBD	Enable internal peripheral bus (60x bus) data parity checking 0 Parity checking is disabled. 1 Allows a data parity error to cause a checkstop if MSR[ME] = 0 or a machine check exception if MSR[ME] = 1 EBA and EBD let the processor operate with memory subsystems that do not generate parity.
4	SBCLK	CKO output enable and clock type selection. When PMCR1[CKO_SEL] = 0, this bit is used in conjunction with HID0[ECLK] and the hard reset signals to configure CKO. See Table 2-2.
5	—	EICE bit on some other PowerPC devices This bit is not used in the MPC8240 (and so it is reserved).
6	ECLK	CKO output enable and clock type selection. When PMCR1[CKO_SEL] = 0, this bit is used in conjunction with HID0[SBCLK] and the hard reset signals to configure CKO. See Table 2-2.
7	—	PAR bit on some other PowerPC devices to disable precharge of $\overline{\text{ARTRY}}$ signal This bit is not used in the MPC8240 (and so it is reserved).
8	DOZE	Doze mode enable. Operates in conjunction with MSR[POW]. ¹ 0 Processor doze mode disabled. 1 Processor 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] ¹ 0 Processor nap mode disabled 1 Processor nap mode enabled. Nap mode is invoked by setting MSR[POW] while this bit is set. When this occurs, the processor indicates that it is ready to enter nap mode. If the peripheral logic determines that the processor may enter nap mode (no more snooping of the internal buffers is required), the processor enters nap mode after several processor clocks. In nap mode, the PLL and the time base remain active. Note that the MPC8240 asserts the $\overline{\text{QACK}}$ output signal depending on the power-saving state of the peripheral logic, and not on the power-saving state of the processor core.
10	SLEEP	Sleep mode enable—Operates in conjunction with MSR[POW] ¹ 0 Processor sleep mode disabled. 1 Processor sleep mode enabled—Sleep mode is invoked by setting MSR[POW] while this bit is set. When this occurs, the processor indicates that it is ready to enter sleep mode. If the peripheral logic determines that the processor may enter sleep mode (no more snooping of the internal buffers is required), 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[0–4] to PLL bypass mode, and then disabling the internal sys_logic-clk signal. Note that the MPC8240 asserts the $\overline{\text{QACK}}$ output signal depending on the power-saving state of the peripheral logic, and not on the power-saving state of the processor core.
11	DPM	Dynamic power management enable ¹ 0 Processor 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.
12–14	—	Reserved

Table 2-1. HID0 Field Descriptions (Continued)

Bits	Name	Description
15	NHR	Not hard reset (software-use only)—Helps software distinguish a hard reset from a soft reset. 0 A hard reset occurred if software had previously set this bit. 1 A hard reset has not occurred. If software sets this bit after a hard reset, when a reset occurs and this bit remains set, software can tell it was a soft reset.
16	ICE	Instruction cache enable ² 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 accesses are propagated to the bus as single-beat transactions. For these transactions, however, the processor reflects the original state of the I bit (from the MMU) to the peripheral logic block, regardless of cache disabled status. ICE is zero at power-up. 1 The instruction cache is enabled
17	DCE	Data cache enable ² 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 accesses are propagated to the bus as single-beat transactions. For those transactions, however, the processor reflects the original state of the I bit (from the MMU) to the peripheral logic block, regardless of cache disabled status. ICE is zero at power-up. 1 The data cache is enabled.
18	ILOCK	Instruction cache lock 0 Normal operation 1 Instruction cache is locked. A locked cache supplies data normally on a hit, but an access is treated as a cache-inhibited transaction on a miss. On a miss, the transaction to the bus is single-beat. To prevent locking during a cache access, an isync must precede the setting of ILOCK.
19	DLOCK	Data cache lock 0 Normal operation 1 Data cache is locked. A locked cache supplies data normally on a hit but an access is treated as a cache-inhibited transaction on a miss. On a miss, the transaction to the bus is single-beat. 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 must precede the setting of DLOCK.
20	ICFI	Instruction cache flash invalidate ² 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 without writing back modified cache blocks to memory. Cache access is blocked during this time. Accesses to the cache from the peripheral logic bus are signaled as a miss during invalidate-all operations. Setting ICFI clears all the valid bits of the blocks and the PLRU bits to point to way LO of each set. Once this flash invalidate bit is set through an mtspr instruction, hardware automatically resets this bit in the next cycle (provided that the corresponding cache enable bit is set in HID0).

Table 2-1. HID0 Field Descriptions (Continued)

Bits	Name	Description
21	DCFI	Data cache flash invalidate ² 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. Accesses to the cache from the peripheral logic bus are signaled as a miss during invalidate-all operations. Setting DCFI clears all the valid bits of the blocks and the PLRU bits so that they point to way L0 of each set. Once the flash invalidate bit is set through an mtspr instruction, hardware automatically resets this bit in the next cycle (provided that the corresponding cache enable bit is set in HID0).
22–23	—	Reserved
24	—	IFEM bit on some other PowerPC devices This bit is not used in the MPC8240 (and so it is reserved).
25–26	—	Reserved
27	FBIOB	Force branch indirect on bus 0 Register indirect branch targets are fetched normally 1 Forces register indirect branch targets to be fetched externally.
28	—	Reserved—Used as address broadcast enable bit on some other PowerPC devices
29–30	—	Reserved
31	NOOPTI	No-op the data cache touch instructions 0 The dcbt and dcbtst instructions are enabled. 1 The dcbt and dcbtst instructions are no-oped globally.

¹ See Chapter 9, “Power Management,” of the MPC603e User’s Manual for more information.

² See Chapter 3, “Instruction and Data Cache Operation,” of the MPC603e User’s Manual for more information.

Table 2-2 shows how HID0[SBCLK], HID0[ECLK], and the hard reset signals are used to configure CKO when PCMR[CKO_SEL] = 0. When PCMR[CKO_SEL] = 1, the CKO_MODE field of PCMR determines the signal driven on CKO.

Table 2-2. HID0[BCLK] and HID0[ECLK] CKO Signal Configuration

HRST_CPU and HRST_CTRL	HID0[ECLK]	HID0[SBCLK]	Signal Driven on CKO
Asserted	x	x	sys-logic-clk
Negated	0	0	High impedance
Negated	0	1	sys-logic-clk divided by 2
Negated	1	0	Processor core clock
Negated	1	1	sys-logic-clk

2.3.1.2.2 Hardware Implementation-Dependent Register 1 (HID1)

The MPC8240 implementation of HID1 is shown in Figure 2-4.

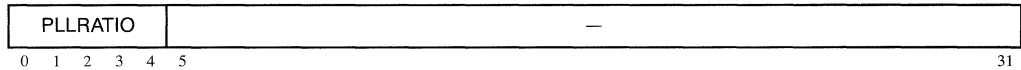


Figure 2-4. Hardware Implementation Register 1 (HID1)

Table 2-3 shows the bit definitions for HID1.

Table 2-3. HID1 Field Descriptions

Bits	Name	Function
0–4	PLLRATIO	PLL configuration processor core frequency ratio—This read-only field is determined by the value on the PLL_CFG[0–4] signals during reset and the processor-to-memory clock frequency ratio defined by that PLL_CFG[0–4] value. See MPC8240 Hardware Specification for a listing of supported settings. Note that multiple settings of the PLL_CFG[0–4] signals can map to the same PLLRATIO value. Thus, system software cannot read the PLLRATIO value and associate it with a unique PLL_CFG[0–4] value.
5–31	—	Reserved

2.3.1.2.3 Hardware Implementation-Dependent Register 2 (HID2)

The processor core implements an additional hardware implementation-dependent register as shown in Figure 2-5, not described in the *MPC603e User's Manual*.

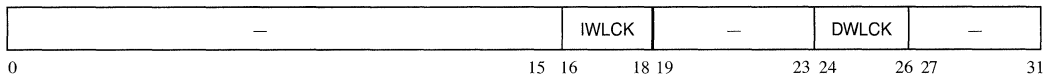


Figure 2-5. Hardware Implementation-Dependent Register 2 (HID2)

Table 2-4 describes the HID2 fields.

Table 2-4. HID2 Field Descriptions

Bits	Name	Function
0–15	—	Reserved
16–18	IWLCK	Instruction cache way lock—Useful for locking blocks of instructions into the instruction cache for time-critical applications where deterministic behavior is required. Refer to Section 2.4.2.3, “Cache Locking,” for more information.
19–23	—	Reserved
24–26	DWLCK	Data cache way lock—Useful for locking blocks of data into the data cache for time-critical applications where deterministic behavior is required. Refer to Section 2.4.2.3, “Cache Locking,” for more information.
27–31	—	Reserved

2.3.1.2.4 Processor Version Register (PVR)

Software can identify the MPC8240's processor core by reading the processor version register (PVR). The MPC8240's processor version number is 0x0081; the processor revision level starts at 0x0100 and is incremented for each revision of the chip. This information is useful for data cache flushing routines for identifying the size of the cache and identifying this processor as one that supports cache locking.

2.3.2 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 greatly simplifies instruction pipelining.

2.3.2.1 Calculating Effective Addresses

The effective address (EA) is the 32-bit address computed by the processor when executing a memory access or branch instruction or when fetching the next sequential instruction.

The PowerPC architecture supports two simple memory addressing modes:

- $EA = (rA)0 + \text{offset}$ (including offset = 0) (register indirect with immediate index)
- $EA = (rA)0 + rB$ (register indirect with index)

These simple addressing modes allow efficient address generation for memory accesses. Calculation of the effective address for aligned transfers occurs in a single clock cycle.

For a memory access instruction, if the sum of the effective address and the operand length exceeds the maximum effective address, the memory operand is considered to wrap around from the maximum effective address to effective address 0.

Effective address computations for both data and instruction accesses use 32-bit unsigned binary arithmetic. A carry from bit 0 is ignored in 32-bit implementations.

In addition to the functionality of the MPC603e, the MPC8240 has additional hardware support for misaligned little-endian accesses. Except for string/multiple load and store instructions, little-endian load/store accesses not on a word boundary generate exceptions under the same circumstances as big-endian requests.

2.3.2.2 PowerPC Instruction Set

The PowerPC instructions are divided into the following categories:

- Integer instructions—These include computational and logical instructions.
 - Integer arithmetic — divide instructions execute with a shorter latency as described in Section 2.7, “Instruction Timing.”
 - Integer compare
 - Integer logical
 - Integer rotate and shift

- Floating-point instructions—These include floating-point computational instructions, as well as instructions that affect the FPSCR.
 - Floating-point arithmetic
 - Floating-point multiply/add
 - Floating-point rounding and conversion
 - Floating-point compare
 - Floating-point status and control
- Load/store instructions—These include integer and floating-point load and store instructions.
 - Integer load and store
 - Integer load and store with byte reverse
 - Integer load and store string/multiple
 - Floating-point load and store
- Flow control instructions—These include branching instructions, condition register logical instructions, trap instructions, and other synchronizing instructions that affect the instruction flow.
 - Branch and trap
 - Condition register logical
 - Primitives used to construct atomic memory operations (**lwarx** and **stwx**.)
 - Synchronize
- Processor control instructions—These instructions are used for synchronizing memory accesses and management of caches, TLBs, and the segment registers.
 - Move to/from SPR
 - Move to/from MSR
 - Instruction synchronize
- Memory control instructions—These provide control of caches, TLBs, and segment registers.
 - Supervisor-level cache management
 - User-level cache management
 - Segment register manipulation
 - TLB management

Note that this grouping of the instructions does not indicate which execution unit executes a particular instruction or group of instructions.

Integer instructions operate on byte, half-word, and word operands. The PowerPC architecture uses instructions that are four 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.

Cache Implementation

Floating-point instructions operate on single-precision (one word) and double-precision (one double word) floating-point operands. It also provides for word and double-word operand loads and stores between memory and a set of 32 floating-point registers (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 separate instructions. Decoupling computational instructions from memory accesses increases throughput by facilitating pipelining.

PowerPC processors follow the program flow when they are 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 exception may cause one of several components of the system software to be invoked.

2.3.2.3 MPC8240 Implementation-Specific Instruction Set

The MPC8240 processor core instruction set is defined as follows:

- The processor core provides hardware support for all 32-bit PowerPC instructions.
- The processor 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 processor 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 Indexed (**stfiwx**)
 - External Control In Word Indexed (**eciwx**)
 - External Control Out Word Indexed (**ecowx**)

The MPC8240 does not provide the hardware support for misaligned **eciwx** and **ecowx** instructions provided by the MPC603e processor. An alignment exception is taken if these instructions are not word-aligned.

2.4 Cache Implementation

The MPC8240 processor core has separate data and instruction caches. The cache implementation is described in the following sections.

2.4.1 PowerPC Cache Model

The PowerPC architecture does not define hardware aspects of cache implementations. For example, some PowerPC processors, including the MPC8240's processor core, have separate instruction and data caches (Harvard architecture); others, such as the PowerPC 601® microprocessor implement a unified cache.

PowerPC microprocessors control the following memory access modes on a page or block basis:

- Write-back/write-through mode
- Caching-inhibited mode
- Memory coherency

The PowerPC cache management instructions provide a means by which the application programmer can affect the cache contents.

2.4.2 MPC8240 Implementation-Specific Cache Implementation

As shown in Figure 2-1, the caches provide a 64-bit interface to the instruction fetch unit and load/store unit. 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.

Each cache block contains eight contiguous words from memory that are loaded from an 8-word boundary (that is, bits A27–A31 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 cache blocks are loaded in to the processor core in four beats of 64 bits each. The burst load is performed as critical double word first.

To ensure coherency among caches in a multiprocessor (or multiple caching-device) implementation, the processor core implements the MEI protocol. These three states, modified, exclusive, and invalid, indicate the state of the cache block as follows:

- 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.
- Invalid—This cache block does not hold valid data.

2.4.2.1 Data Cache

As shown in Figure 2-6, the data cache is configured as 128 sets of four blocks each. Each block consists of 32 bytes, two 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 MPC8240's processor core, the block size is equivalent to a cache line.

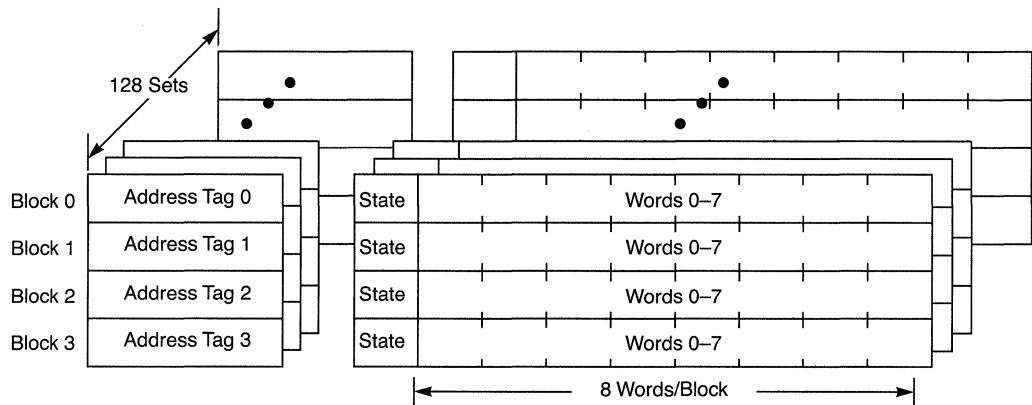


Figure 2-6. Data Cache Organization

Because the processor core data cache tags are single-ported, simultaneous load or 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 which case the snoop is retried and must re-arbitrate for access to the cache. Loads or stores that are deferred due to snoop accesses are executed on the clock cycle following the snoop.

Because the caches on the processor core are write-back caches, the predominant type of transaction to the memory subsystem for most applications is burst-read memory operations, followed by burst-write memory operations, and single-beat (noncacheable or write-through) memory read and write operations. When a cache block is filled with a burst read, the critical double word is simultaneously written to the cache and forwarded to the requesting unit, thus minimizing stalls due to load delays.

Additionally, there can 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 line in the cache). Note that all memory subsystem references are performed by the processor core to the internal peripheral logic bus on the MC8240.

The address and data buses of the internal peripheral logic bus operate independently to support pipelining and split transactions during memory accesses. The processor core pipelines its own transactions to a depth of one level.

Typically, memory accesses are weakly ordered—sequences of operations, including load/store string and multiple instructions, do not necessarily complete in the order they begin—maximizing the efficiency of the internal bus without sacrificing coherency of the data. The processor core allows pending read operations to precede previous store operations (except when a dependency exists, or in cases where a non-cacheable 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.

2.4.2.2 Instruction Cache

The instruction cache also consists of 128 sets of four blocks—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 caused by a cache miss. In the processor core, internal access to the instruction cache is blocked only until the critical load completes.

The processor core supports instruction fetching from other instruction cache lines following the forwarding of the critical first double word of a cache line load operation. The processor core's instruction cache is blocked only until the critical load completes (hits under reloads allowed). Successive instruction fetches from the cache line being loaded are forwarded, and accesses to other instruction cache lines can proceed during the cache line load operation.

The instruction cache is not snooped, and cache coherency must be maintained by software. A fast hardware invalidation capability is provided to support cache maintenance. The organization of the instruction cache is very similar to the data cache shown in Figure 2-6.

2.4.2.3 Cache Locking

The processor core supports cache locking, which is the ability to prevent some or all of a microprocessor's instruction or data cache from being overwritten. Cache entries can be locked for either an entire cache or for individual ways within the cache. Entire data cache locking is enabled by setting `HID0[DLOCK]`, and entire instruction cache locking is enabled by setting `HID0[ILOCK]`. For more information, refer to Cache Locking on the G2 Core application note (order number: AN1767/D). Cache way locking is controlled by the `IWLCK` and `DWLCK` bits of `HID2`.

2.4.2.3.1 Entire Cache Locking

When an entire cache is locked, hits within the cache are supplied in the same manner as hits to an unlocked cache. Any access that misses in the cache is treated as a cache-inhibited access. Cache entries that are invalid at the time of locking will remain invalid and inaccessible until the cache is unlocked. Once the cache has been unlocked, all entries (including invalid entries) are available. Entire cache locking is inefficient if the number of instructions or the size of data to be locked is small compared to the cache size.

2.4.2.3.2 Way Locking

Locking only a portion of the cache is accomplished by locking ways within the cache. Locking always begins with the first way (way0) and is sequential. That is, it is valid to lock ways 0, 1, and 2 but it is not possible to lock just way0 and way2. When using way locking at least one way must be left unlocked. The maximum number of lockable ways is three.

Unlike entire cache locking, invalid entries in a locked way are accessible and available for data placement. As hits to the cache fill invalid entries within a locked way, the entries become valid and locked. This behavior differs from entire cache locking where nothing is placed in the cache, even if invalid entries exist in the cache. Unlocked ways of the cache behave normally.

2.4.3 Cache Coherency

The central control unit (CCU) manages the cache coherency within the MPC8240. It responds to all accesses generated by the processor core and causes the snooping of the addresses in the internal buffers as necessary. Also, the CCU generates snoop transactions on the peripheral logic bus to allow the processor to snoop accesses between the PCI interface and memory. Refer to Chapter 7, “Central Control Unit,” for more detailed information about the internal address and data buffers in the MPC8240.

2.4.3.1 CCU Responses to Processor Transactions

The processor core generates various types of read and write accesses as well as address-only transactions. Table 2-5 shows all the types of internal transactions performed by the processor core and the CCU responses.

Table 2-5. CCU Responses to Processor Transactions

Processor Transaction	CCU Response
Read	Directs read to appropriate interface
Read-with-intent-to-modify	Directs read to appropriate interface
Read atomic	Directs read to appropriate interface
Read-with-intent-to-modify-atomic	Directs read to appropriate interface
Write-with-flush	Directs write to appropriate interface
Write-with-kill	Directs write to appropriate interface
Write-with-flush-atomic	Directs write to appropriate interface
sync	CCU buffers are flushed.
eieio	CCU buffers are flushed.
Kill block (generated by dcbz instruction when the addressed block has either the E or M bits set)	CCU buffers are snooped.
icbi	CCU buffers are snooped.
Read-with-no-intent-to-cache	Directs read to appropriate interface
Clean	CCU takes no further action.
Flush	CCU takes no further action.
tlbie	CCU takes no further action.
lwarx , reservation set	CCU takes no further action. (The MPC8240 does not support atomic references in PCI memory space.)
stwcx., reservation set	CCU takes no further action. (The MPC8240 does not support atomic references in PCI memory space.)
tlbsync	CCU takes no further action.

Table 2-5. CCU Responses to Processor Transactions (Continued)

Processor Transaction	CCU Response
Graphic write (ecowx)	Processor transaction error. Machine check signalled to processor core (if enabled)
Graphic read (eciwx)	Processor transaction error. Machine check signalled to processor core (if enabled)

2.4.3.2 Processor Responses to PCI-to-Memory Transactions

The CCU controls the data flow between the PCI interface and the memory interface. One of its functions is to broadcast these transactions on the peripheral logic bus so that the processor core can snoop the L1 cache as needed (if snooping is enabled). Table 2-5 shows all the types of transactions reflected by the CCU to the processor core for snooping.

Table 2-6. Transactions Reflected to the Processor for Snooping

Snooped Transaction	Condition Detected by CCU	Processor Response
Read	Non-locked PCI read from memory	All burst reads observed on the bus are snooped as if they were writes, causing the addressed cache block to be flushed. A read marked as global causes the following responses: <ul style="list-style-type: none"> • If the addressed block in the cache is invalid, the processor takes no action. • If the addressed block in the cache is in the exclusive state, the block is invalidated. • If the addressed block in the cache is in the modified state, the block is flushed to memory and the block is invalidated.
Read-with-intent-to-modify (RWITM)-atomic	Locked PCI read from memory	A RWITM operation is issued to acquire exclusive use of a memory location for the purpose of modifying it. <ul style="list-style-type: none"> • If the addressed block is invalid, the processor takes no action. • If the addressed block in the cache is in the exclusive state, the processor changes the state of the cache block to invalid. • If the addressed block in the cache is in the modified state, the block is flushed to memory and the block is invalidated.
Write-with-flush Write-with-flush-atomic	Non-locked PCI write to memory or locked PCI write to memory, respectively	<ul style="list-style-type: none"> • If the addressed block is in the exclusive state, the snoop forces the state of the addressed block to invalid. • If the addressed block is in the modified state, the snoop causes a push of the modified block out of the cache to memory and changes the state of the block to invalid.
Write-with-kill	Locked or non-locked PCI write with invalidate to memory	In a write-with-kill operation, the processor snoops the cache for a copy of the addressed block. If one is found the cache block is forced to the I state, killing modified data that may have been in the block.

2.5 Exception Model

This section describes the PowerPC exception model and implementation-specific details of the MPC8240 core.

2.5.1 PowerPC Exception Model

The PowerPC exception mechanism allows the processor to change to supervisor state as a result of external signals, errors, or unusual conditions arising in the execution of instructions. When exceptions occur, information about the state of the processor is saved to certain registers and the processor begins execution at an address (exception vector) predetermined for each exception. Processing of exceptions occurs in supervisor mode.

Although multiple exception conditions can map to a single exception vector, a more specific condition may be determined by examining a register associated with the exception. For example, the DSISR identifies instructions that cause a DSI exception. Additionally, some exception conditions can be explicitly enabled or disabled by software.

The PowerPC architecture requires that exceptions be handled in program order; therefore, although a particular implementation may recognize exception conditions out of order, exceptions are taken in strict order. When an instruction-caused exception 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 exception is taken. Any exceptions caused by those instructions are handled first. Likewise, exceptions that are asynchronous and precise are recognized when they occur, but are not handled until the instruction currently in the completion stage successfully completes execution or generates an exception, and the completed store queue is emptied.

Unless a catastrophic condition causes a system reset or machine check exception, only one exception is handled at a time. If, for example, a single instruction encounters multiple exception conditions, those conditions are handled sequentially. After the exception handler handles an exception, the instruction execution continues until the next exception condition is encountered. However, in many cases there is no attempt to re-execute the instruction. This method of recognizing and handling exception conditions sequentially guarantees that exceptions are recoverable.

Exception handlers should save the information stored in SRR0 and SRR1 early to prevent the program state from being lost due to a system reset or machine check exception or to an instruction-caused exception in the exception handler. SRR0 and SRR1 should also be saved before enabling external interrupts.

The PowerPC architecture supports four types of exceptions:

- Synchronous, precise—These are caused by instructions. All instruction-caused exceptions are handled precisely; that is, the machine state at the time the exception occurs is known and can be completely restored. This means that (excluding the trap and system call exceptions) the address of the faulting instruction is provided to the exception handler and neither the faulting instruction nor subsequent instructions in the code stream will complete execution before the exception is taken. Once the exception is processed, execution resumes at the address of the faulting instruction (or at an alternate address provided by the exception handler). When an exception 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. These are not implemented on the MPC8240.
- Asynchronous, maskable—The external, system management interrupt (SMI), and decremter interrupts are maskable asynchronous exceptions. When these exceptions occur, their handling is postponed until the next instruction and any

Exception Model

exceptions associated with that instruction complete execution. If no instructions are in the execution units, the exception is taken immediately upon determination of the correct restart address (for loading SRR0).

- Asynchronous, nonmaskable—There are two nonmaskable asynchronous exceptions— system reset and the machine check exception. These exceptions may not be recoverable, or may provide a limited degree of recoverability. All exceptions report recoverability through MSR[RI].

2.5.2 MPC8240 Implementation-Specific Exception Model

As specified by the PowerPC architecture, all processor core exceptions can be described as either precise or imprecise and either synchronous or asynchronous. Asynchronous exceptions (some of which are maskable) are caused by events external to the processor's execution. Synchronous exceptions, which are all handled precisely by the processor core, are caused by instructions. The processor core exception classes are shown in Table 2-7.

Table 2-7. Exception Classifications for the Processor Core

Synchronous/Asynchronous	Precise/Imprecise	Exception Type
Asynchronous, nonmaskable	Imprecise	Machine check System reset
Asynchronous, maskable	Precise	External interrupt Decrementer System management interrupt
Synchronous	Precise	Instruction-caused exceptions

Although exceptions have other characteristics as well, such as whether they are maskable or nonmaskable, the distinctions shown in Table 2-7 define categories of exceptions that the processor core handles uniquely. Note that Table 2-7 includes no synchronous imprecise instructions.

The processor core's exceptions, and conditions that cause them, are listed in Table 2-8.

Table 2-8. Exceptions and Conditions

Exception Type	Vector Offset (hex)	Causing Conditions
Reserved	00000	—
System reset	00100	A system reset is caused by the assertion of HRST_CPU, SRESET or sreset (asserted by the EPIC unit)
Machine check	00200	A machine check exception is caused by the assertion of the NMI input signal or the occurrence of internal errors as described in Chapter 13, "Error Handling and Exceptions." This exception occurs when a machine check condition is detected, the error is enabled, HID0[EMCP] is set, PICR1[MCP_EN] is set, and MSR[ME] is set. When one of these errors occurs, the MPC8240 takes the exception and asserts the MCP output signal.

Table 2-8. Exceptions and Conditions (Continued)

Exception Type	Vector Offset (hex)	Causing Conditions
DSI	00300	<p>The cause of a DSI exception can be determined by the bit settings in the DSISR, listed as follows:</p> <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), 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. 5 Set by an eciwX or ecowX instruction if the access is to an address that is marked as write-through or execution of a load/store instruction that accesses a direct-store segment. 6 Set for a store operation and cleared for a load operation. 11 Set if eciwX or ecowX is used and EAR[E] is cleared.
ISI	00400	<p>An ISI exception is caused when an instruction fetch cannot be performed for any of the following reasons:</p> <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 exception must be taken to load the PTE (and possibly the page) into memory. • The fetch access is to a direct-store segment (indicated by SRR1[3] set). • The fetch access violates memory protection (indicated by SRR1[4] set). If the key bits (Ks and Kp) in the segment register and the PP bits in the PTE are set to prohibit read access, instructions cannot be fetched from this location.
External interrupt	00500	<p>An external interrupt is caused when MSR[EE] = 1 and the internal $\overline{\text{int}}$ signal is asserted by the EPIC interrupt module to the processor core.</p>
Alignment	00600	<p>An alignment exception is caused when the processor core cannot perform a memory access for any of the reasons described below:</p> <ul style="list-style-type: none"> • The operand of a floating-point load or store is to a direct-store segment. • The operand of a floating-point load or store is not word-aligned. • The operand of a lmw, stmw, lwarx, or stwcx is not word-aligned. • The operand of an elementary, multiple or string load or store crosses a segment boundary with a change to the direct store T bit. • The operand of dcbz instruction is in memory that is write-through required or caching inhibited, or dcbz is executed in an implementation that has either no data cache or a write-through data cache. • A misaligned eciwX or ecowX instruction • A multiple or string access with MSR[LE] set <p>The processor core differs from MPC603e User's Manual in that it initiates an alignment exception when it detects a misaligned eciwX or ecowX instruction and does not initiate an alignment exception when a little-endian access is misaligned.</p>

Table 2-8. Exceptions and Conditions (Continued)

Exception Type	Vector Offset (hex)	Causing Conditions
Program	00700	A program exception is caused by one of the following exception conditions, which correspond to bit settings in SRR1 and arise during execution of an instruction: <ul style="list-style-type: none"> Illegal instruction—An illegal instruction program exception 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 processor core), or when execution of an optional instruction not provided in the processor core is attempted (these do not include those optional instructions that are treated as no-ops). Privileged instruction—A privileged instruction type program exception is generated when the execution of a privileged instruction is attempted and the MSR register user privilege bit, MSR[PR], is set. In the processor core, this exception is generated for mtspr or mfspr with an invalid SPR field if SPR[0] = 1 and MSR[PR] = 1. This may not be true for all PowerPC processors. Trap—A trap type program exception is generated when any of the conditions specified in a trap instruction are met.
Floating-point unavailable	00800	A floating-point unavailable exception is caused by an attempt to execute a floating-point instruction (including floating-point load, store, and move instructions) when the floating-point available bit is cleared (MSR[FP] = 0).
Decrementer	00900	The decrementer exception occurs when the most significant bit of the decrementer (DEC) register transitions from 0 to 1. Must also be enabled with the MSR[EE] bit.
Reserved	00A00–00BFF	—
System call	00C00	A system call exception occurs when a System Call (sc) instruction is executed.
Trace	00D00	A trace exception is taken when MSR[SE] = 1 or when the currently completing instruction is a branch and MSR[BE] = 1.
Floating-point assist	00E00	The MPC8420 does not generate an exception to this vector. Other PowerPC processors may use this vector for floating-point assist exceptions.
Reserved	00E10–00FFF	—
Instruction translation miss	01000	An instruction translation miss exception is caused when the effective address for an instruction fetch cannot be translated by the ITLB.
Data load translation miss	01100	A data load translation miss exception is caused when the effective address for a data load operation cannot be translated by the DTLB.
Data store translation miss	01200	A data store translation miss exception is caused when the effective address for a data store operation cannot be translated by the DTLB, or when a DTLB hit occurs, and the changed bit in the PTE must be set due to a data store operation.
Instruction address breakpoint	01300	An instruction address breakpoint exception occurs when the address (bits 0–29) in the IABR matches the next instruction to complete in the completion unit, and the IABR enable bit (bit 30) is set.

Table 2-8. Exceptions and Conditions (Continued)

Exception Type	Vector Offset (hex)	Causing Conditions
System management interrupt	01400	A system management interrupt is caused when MSR[EE] = 1 and the SMI input signal is asserted.
Reserved	01500–02FFF	—

2.5.3 Exception Priorities

The exception priorities for the processor core are unchanged from those described in the MPC603e User’s Manual except for the alignment exception, whose causes are prioritized as follows:

1. Floating-point operand not word-aligned
2. **lmw**, **stmw**, **lwarx**, or **stwcx** operand not word-aligned
3. **eciwx** or **ecowx** operand misaligned
4. A multiple or string access is attempted with MSR[LE] set.

Also, there is a priority mechanism for all the conditions specific to the MPC8240 that can cause a machine check exception. These are described in Chapter 13, “Error Handling and Exceptions.”

2.6 Memory Management

The following subsections describe the memory management features of the PowerPC architecture and the MPC8240 implementation.

2.6.1 PowerPC MMU Model

The primary functions of the MMU are:

- to translate logical (effective) addresses to physical addresses for memory accesses
- to provide access protection on blocks and pages of memory

There are two types of accesses generated by the processor core that require address translation—instruction accesses and data accesses to memory generated by load and store instructions.

The PowerPC MMU and exception models 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 PowerPC architecture supports the following three translation methods:

- Address translations disabled. Translation is enabled by setting bits in the MSR — MSR[IR] enables instruction address translations and MSR[DR] enables data address translations. Clearing these bits disables translation and the effective address is used as the physical address.
- Block address translation. The PowerPC architecture defines independent four-entry BAT arrays for instructions and data that maintain address translations for blocks of memory. Block sizes range from 128 Kbyte to 256 Mbyte and are software selectable. The BAT arrays are maintained by system software. The BAT registers, defined by the PowerPC architecture for block address translations, are shown in Figure 2-2.
- Demand page mode. The page table contains a number of page table entry groups (PTEGs). A PTEG contains eight page table entries (PTEs) of eight bytes each; therefore, each PTEG is 64 bytes long. PTEG addresses are entry points for table search operations.

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 2; its starting address is a multiple of its size.

On-chip 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. In the MPC8240, the processor core's TLBs are 64-entry, two-way set-associative caches that contain instruction and data address translations. The MPC8240's core provides hardware assist for software table search operations through the hashed page table on TLB misses. Supervisor software can invalidate TLB entries selectively.

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.

2.6.2 MPC8240 Implementation-Specific MMU Features

The instruction and data MMUs in the processor 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.

The MPC8240 MMUs support up to 4 Petabytes (2^{52}) of virtual memory and 4 Gbytes (2^{32}) of physical memory (referred to as real memory in the PowerPC architecture specification) for instructions and data. Referenced and changed status is maintained by the processor for each page to assist implementation of a demand-paged virtual memory system.

The MPC8240 TLBs are 64-entry, two-way set-associative caches that contain instruction and data address translations. The processor core provides hardware assist for software

table search operations through the hashed page table on TLB misses. Supervisor software can invalidate TLB entries selectively.

After an effective address is generated, the higher-order bits of the effective address are translated by the appropriate MMU into physical address bits. Simultaneously, the lower-order address bits (that are untranslated; therefore, considered both logical and physical), are directed to the on-chip caches where they form the index into the four-way set-associative tag array. After translating the address, the MMU passes the higher-order bits of the physical address 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 system interface, which accesses external memory.

For instruction accesses, the MMU performs an address lookup in both the 64 entries of the ITLB, and in the IBAT array. If an effective address hits in both the ITLB and the IBAT array, the IBAT array translation takes priority. Data accesses cause a lookup in the DTLB and DBAT array for the physical address translation. In most cases, the physical address translation resides in one of the TLBs and the physical address bits are readily available to the on-chip cache.

When the physical address translation misses in the TLBs, the processor core provides hardware assistance for software to search the translation tables in memory. When a required TLB entry is not found in the appropriate TLB, the processor vectors to one of the three TLB miss exception handlers so that the software can perform a table search operation and load the TLB. When this occurs, the processor automatically saves information about the access and the executing context. Refer to the MPC603e User's Manual for more detailed information about these features and the suggested software routines for searching the page tables.

2.7 Instruction Timing

The processor core is a pipelined superscalar processor. A pipelined processor is one in which the processing of an instruction is broken into discrete stages. Because the processing of an instruction is broken into a series of stages, an instruction does not require the entire resources of an execution unit at one 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. The instruction pipeline in the processor core has four major 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, the BPU decodes branches during the fetch stage and folds out branch instructions before the dispatch stage if possible.

- 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.
- During the execute pipeline stage, each execution unit that has an executable instruction executes the selected instruction (perhaps over multiple cycles), writes the instruction's result into the appropriate rename register, and notifies the completion stage that the instruction has finished execution. In the case of an internal exception, the execution unit reports the exception to the completion/writeback pipeline stage and discontinues instruction execution until the exception is handled. The exception is not signaled until that instruction is the next to be completed. Execution of most load/store instructions is also pipelined. The load/store unit has two pipeline stages. The first stage is for effective address calculation and MMU translation, and the second stage is for accessing the data in the cache.
- The complete/writeback 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 exception, all following instructions are cancelled, their execution results in rename registers are discarded, and instructions are fetched from the correct instruction stream.

The processor core provides support for single-cycle store operations and provides an adder/comparator in the SRU that allows the dispatch and execution of multiple integer add and compare instructions on each cycle.

Performance of integer divide operations has been improved in the processor core. Execution of a divide instruction takes half the cycles to execute than that described in the MPC603e User's Manual. The new latency is reflected in Table 2-9.

Table 2-9. Integer Divide Latency

Primary Opcode	Extended Opcode	Mnemonic	Form	Unit	Cycles
31	459	divwu[o][.]	xo	IU	20
31	491	divw[o][.]	xo	IU	20

2.8 Differences between the MPC8240 Core and the PowerPC 603e Microprocessor

The MPC8240 processor core is a derivative of the MPC603e microprocessor design. Some changes have been made and are visible either to a programmer or a system designer. Any software designed around an MPC603e is functional when replaced with the MPC8240 except for the specific customer-visible changes listed in Table 2-10.

Software can distinguish between the MPC603e and the MPC8240 by reading the processor version register (PVR). The MPC8240's processor version number is 0x0081; the processor revision level starts at 0x0100 and is incremented for each revision of the chip. It is expected that this information is most useful for programmers writing data cache flush routines.

Table 2-10. Major Differences between MPC8240's Core and the MPC603e User's Manual

Description	Impact
Changed HID1 to add bus frequency multipliers as described in the MPC8240 Hardware Specification	On extra bit is provided for PLL configuration, and some other unused encodings of the PLL_CFG[0–4] are now defined.
Added hardware support for misaligned little endian accesses	Except for strings/multiples, little-endian load/store accesses not on a word boundary generate exceptions under the same circumstances as big-endian accesses.
Removed misalignment support for eciwx and ecowx instructions.	These instructions cause an alignment exception if the operands are not on a word boundary.
Removed HID0[5]; now reserved	There is no support for ICE pipeline tracking.
Removed HID0[7]; now reserved	No impact, as the MPC8240 has no $\overline{\text{ARTRY}}$ signal.
Added instruction and data cache locking mechanism	Implements a cache way locking mechanism for both the instruction and data caches. One to three of the four ways in the cache can be locked with control bits in the HID2 register. See Section 2.3.1.2.3, "Hardware Implementation-Dependent Register 2 (HID2)."
Improved access to cache during block fills	The MPC8240 provides quicker access to incoming data and instruction on a cache block fill. See Section 2.4.2, "MPC8240 Implementation-Specific Cache Implementation."
Improved integer divide latency	Performance of integer divide operations has been improved in the processor core. A divide takes half the cycles to execute as described in MPC603e User's Manual. The new latency is reflected in Table 2-9.
No support for dcbz instruction in areas of memory that are write-through and can be accessed by multiple logical addresses	This was previously documented as an anomaly in the MPC603e. Stores of zeros must be used instead of the dcbz instruction when the memory area is designated as write-through, coherency required.
Areas of memory accessed by dcbz instruction should not be marked as global	This was previously documented as an anomaly in the MPC603e. Areas of memory accessed by a dcbz instruction must be marked as not global in the BAT or PTE.

Chapter 3

Signal Descriptions and Clocking

This chapter provides descriptions of the MPC8240's external signals. It describes each signal's behavior when the signal is asserted and negated and when the signal is an input or an output.

NOTE

A bar over a signal name indicates that the signal is active low—for example, \overline{AS} (address strobe). Active-low signals are referred to as asserted (active) when they are low and negated when they are high. Signals that are not active low, such as NMI (nonmaskable interrupt), are referred to as asserted when they are high and negated when they are low.

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

The chapter is organized into the following sections:

- Overview of signals and complete cross-reference for signals that serve multiple functions. Includes listing of output signal states at reset.
- Signal description section that provides a detailed description of each signal, listed by functional block.
- A complete section on the operation of the many input and output clock signals on the MPC8240, and the interactions between these signals.
- A listing of the reset configuration signals and the modes they define.

3.1 Signal Overview

The MPC8240's signals are grouped as follows:

- PCI interface signals
- Memory interface signals
- EPIC control signals

Signal Overview

- I²C interface signals
- System control, power management, and debug signals
- Test/configuration signals
- Clock signals

Figure 3-1 illustrates the external signals of the MPC8240, showing how the signals are grouped. Refer to the *MPC8240 Hardware Specification* for a pinout diagram showing actual pin numbers and including all of the electrical and mechanical specifications.

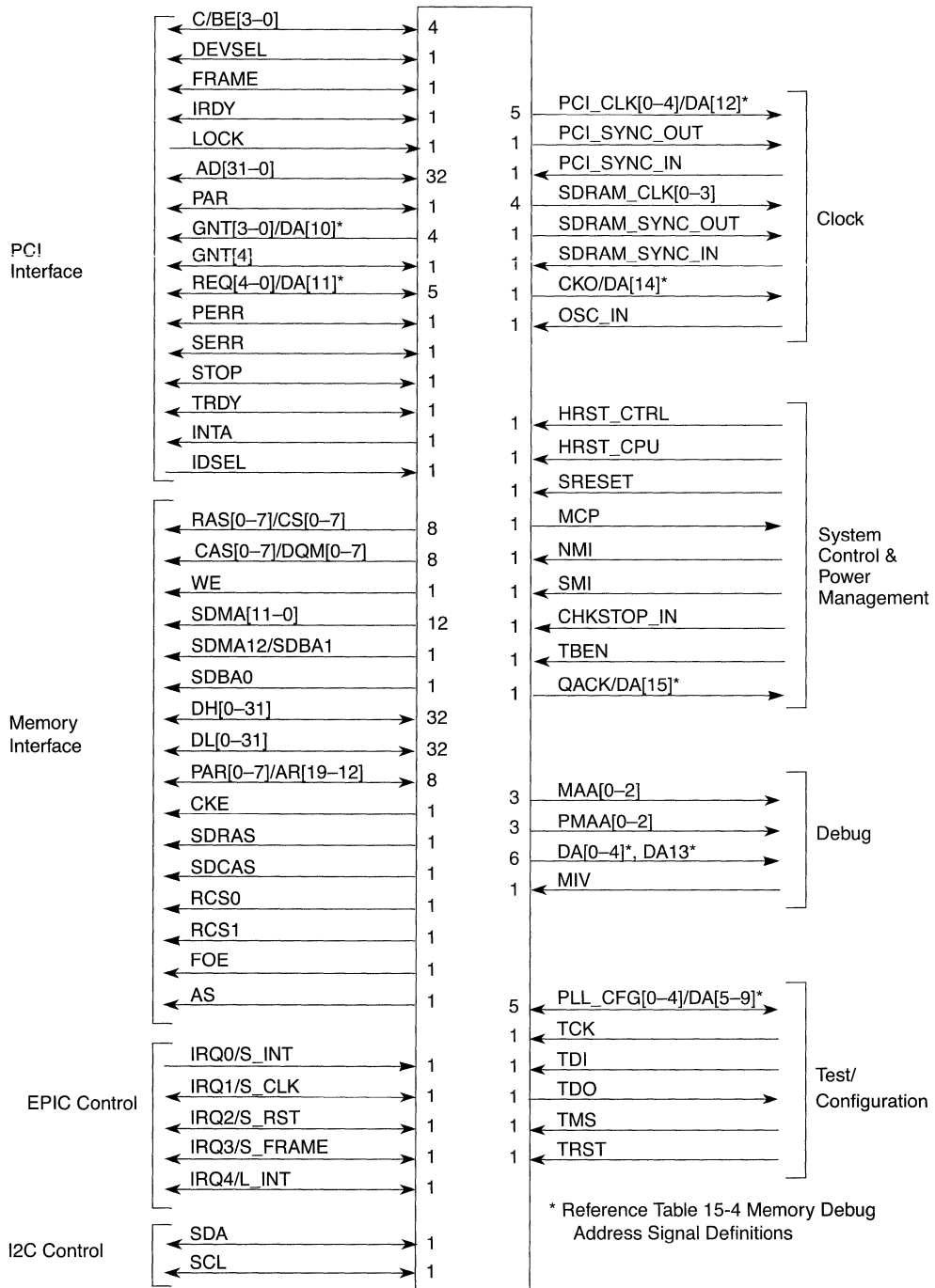


Figure 3-1. MPC8240 Signal Groupings

3.1.1 Signal Cross Reference

The following sections are intended to provide a quick summary of signal functions. Table 3-1 provides an alphabetical cross-reference to the signals of the MPC8240. It details the signal name, interface, alternate functions, number of signals, whether the signal is an input, output, or bidirectional, and finally a pointer to the section in this chapter where the signal is described.

Table 3-1. MPC8240 Signal Cross Reference

Signal	Signal Name	Interface	Alternate Function(s)	Pins	I/O	Section #
AD[31–0]	Address/data	PCI	—	32	I/O	3.2.1.1
AR[19–12]	ROM address 19–12	Memory	PAR[0–7]	8	O	3.2.2.12
\overline{AS}^1	Address strobe	Memory	—	1	O	3.2.2.19
\overline{CAS} [0–7]	Column address strobe 0–7	Memory	DQM[0–7]	8	O	3.2.2.2
$\overline{C/BE}$ [3–0]	Command/byte enable	PCI	—	4	I/O	3.2.1.2
$\overline{CHKSTOP_IN}$	Checkstop in	System Control	—	1	I	3.2.5.6
CKE ¹	SDRAM clock enable	Memory	—	1	O	3.2.2.13
CKO	Debug clock	Clock	DA14	1	O	3.2.7.8
\overline{CS} [0–7]	SDRAM chip select	Memory	\overline{RAS} [0–7]	8	O	3.2.2.3
DA[15–11], DA2	Debug addr [15–11, 2]	Debug	—	6	O	3.2.5.9.3
DA[10–6]	Debug addr [10–6]	Debug	PLL_CFG[0–4]	5	O	3.2.5.9.3
DA5 DA4 DA3 DA1 DA0	Debug addr 5 Debug addr 4 Debug addr 3 Debug addr 1 Debug addr 0	Debug	$\overline{GNT4}$ REQ4 PCI_CLK4 CKO \overline{QACK}	5	O	3.2.5.9.3
\overline{DEVSEL}	Device select	PCI	—	1	I/O	3.2.1.3
DH[0–31]	Data bus high	Memory	—	32	I/O	3.2.2.10
DL[0–31] DL[0] ¹	Data bus low	Memory	—	32	I/O	3.2.2.10
DQM[0–7]	SDRAM data qualifier	Memory	\overline{CAS} [0–7]	8	O	3.2.2.4
FOE ¹	Flash output enable	Memory	—	1	O	3.2.2.18
\overline{FRAME}	Frame	PCI	—	1	I/O	3.2.1.4
\overline{GNT} [4–0] $\overline{GNT4/DA5}^1$	PCI bus grant	PCI	$\overline{GNT0}$: PCI bus request $\overline{GNT4}$: DA5	5	O	3.2.1.5
HRST_CPU	Hard reset (processor)	System Control	—	1	I	3.2.5.1.1
$\overline{HRST_CTRL}$	Hard reset (peripheral logic)	System Control	—	1	I	3.2.5.1.2

Table 3-1. MPC8240 Signal Cross Reference (Continued)

Signal	Signal Name	Interface	Alternate Function(s)	Pins	I/O	Section #
IDSEL	ID select	PCI	—	1	I	3.2.1.15
INTA	Interrupt request	PCI	—	1	O	3.2.1.14
IRDY	Initiator ready	PCI	—	1	I/O	3.2.1.7
IRQ0	Interrupt 0	EPIC Control	S_INT	1	I	3.2.3.1
IRQ1	Interrupt 1	EPIC Control	S_CLK	1	I/O	3.2.3.1
IRQ2	Interrupt 2	EPIC Control	S_RST	1	I/O	3.2.3.1
IRQ3	Interrupt 3	EPIC Control	S_FRAME	1	I/O	3.2.3.1
IRQ4	Interrupt 4	EPIC Control	L_INT	1	I/O	3.2.3.1
L_INT	Local interrupt	EPIC Control	IRQ4	1	I/O	3.2.3.3
LOCK	Lock	PCI	—	1	I	3.2.1.8
MAA[0–2] ¹	Memory addr attributes	Debug	—	3	O	3.2.5.9.1
MCP ¹	Machine check	System Control	—	1	O	3.2.5.3
MIV	Memory interface valid	Debug	—	1	I	3.2.5.9.4
NMI	Nonmaskable interrupt	System Control	—	1	I	3.2.5.4
OSC_IN	System clock input	Clock	—	1	I	3.2.7.1
PAR	Parity	PCI	—	1	I/O	3.2.1.9
PAR[0–7]	Data parity 0–7	Memory	AR[19–12]	8	I/O	3.2.2.11
PCI_CLK[0–4] PCI_CLK4/DA12	PCI clock outputs	Clock	PCI_CLK4: DA12	5	O	3.2.7.2
PCI_SYNC_OUT	PCI clock output	Clock	—	1	O	3.2.7.3
PCI_SYNC_IN	PCI clock input	Clock	—	1	I	3.2.7.4
PERR	Parity error	PCI	—	1	I/O	3.2.1.10
PLL_CFG[0–4] ¹	PLL configuration	Test/ Configuration	DA[5–9]	5	I	3.2.6.1
PMAA[0–2] ¹	PCI addr. attributes	Debug	—	3	O	3.2.5.9.2
QACK ¹	Quiesce acknowledge	Power Management	DA15	1	O	3.2.5.8
RAS[0–7]	Row address strobe 0–7	Memory	CS[0–7]	8	O	3.2.2.1
RCS0 ¹	ROM/bank 0 select	Memory	—	1	O	3.2.2.16
RCS1	ROM/bank 1 select	Memory	—	1	O	3.2.2.17
REQ[4–0] REQ4/DA11	PCI bus request	PCI	REQ0: PCI bus grant REQ4: DA11	5	I O	3.2.1.6
S_CLK	Serial interrupt clock	EPIC Control	IRQ1	1	I/O	3.2.3.2.2

Table 3-1. MPC8240 Signal Cross Reference (Continued)

Signal	Signal Name	Interface	Alternate Function(s)	Pins	I/O	Section #
SCL	Serial clock	I ² C Control	—	1	I/O	3.2.4.2
SDA	Serial data	I ² C Control	—	1	I/O	3.2.4.1
SDBA0	SDRAM bank select 0	Memory	See Table 6-2	1	O	3.2.2.9
SDBA1	SDRAM bank select 1	Memory		1	O	3.2.2.8
SDCAS	SDRAM column access strobe	Memory	—	1	O	3.2.2.15
SDMA12	SDRAM address 12	Memory	See Table 6-2	1	O	3.2.2.7
SDMA[11–0]	SDRAM address 11–0	Memory		12	O	3.2.2.6
SDRAM_CLK[0–3]	SDRAM clock outputs	Clock	—	4	O	3.2.7.5
SDRAM_SYNC_OUT	SDRAM clock output	Clock	—	1	O	3.2.7.6
SDRAM_SYNC_IN	SDRAM feedback clock	Clock	—	1	I	3.2.7.7
SDRAS	SDRAM row address strobe	Memory	—	1	O	3.2.2.14
SERR	System error	PCI	—	1	I/O	3.2.1.11
S_FRAME	Serial interrupt frame	EPIC Control	IRQ3	1	I/O	3.2.3.2.4
S_INT	Serial interrupt stream	EPIC Control	IRQ0	1	I	3.2.3.2.1
SMI	System management interrupt	System Control	—	1	I	3.2.5.5
S_RST	Serial interrupt reset	EPIC Control	IRQ2	1	I/O	3.2.3.2.3
SRESET	Soft reset	System Control	—	1	I	3.2.5.2
STOP	Stop	PCI	—	1	I/O	3.2.1.12
TBEN	Time base enable	System Control	—	1	I	3.2.5.7
TCK	JTAG test clock	Test	—	1	I	3.2.6.2
TDO	JTAG test data output	Test	—	1	O	3.2.6.4
TDI	JTAG test data Input	Test	—	1	I	3.2.6.3
TMS	JTAG test mode select	Test	—	1	I	3.2.6.5
TRDY	Target ready	PCI	—	1	I/O	3.2.1.13
TRST	JTAG test reset	Test	—	1	I	3.2.6.6
WE	Write enable	Memory	—	1	O	3.2.2.5

¹ The MPC8240 samples these signals at the negation of reset to determine the reset configuration. After they are sampled, they assume their normal functions. See Section 3.4, "Configuration Pins Sampled at Reset," for more information about their function during reset.

3.1.2 Output Signal States at Reset

When a system reset is recognized (assertion of $\overline{\text{HRST_CPU}}$ and $\overline{\text{HRST_CTRL}}$), the MPC8240 aborts all current internal and external transactions, and releases all bidirectional I/O signals to a high-impedance state. See Section 13.2.1, “System Reset,” for a complete description of the reset functionality.

There are 19 signals that serve alternate functions as reset configuration input signals during system reset. Their default values and the interpretation of their voltage levels during reset are described in Section 3.4, “Configuration Pins Sampled at Reset.”

On reset, the MPC8240 ignores most input signals (except for PCI_SYNC_IN and the reset configuration signals), and drives most of the output signals to an inactive state. Table 3-2 shows the states of the output-only signals that are not used as reset configuration signals during system reset.

Table 3-2. Output Signal States During System Reset

Interface	Signal	State During System Reset
PCI	$\overline{\text{GNT}}[3-0]$ $\overline{\text{INTA}}$	High impedance
Memory	$\overline{\text{CAS/DQM}}[0-7]$ $\overline{\text{RAS/CS}}[0-7]$ $\overline{\text{RCS1}}$ $\overline{\text{SDRAS}}$ $\overline{\text{SDCAS}}$ $\overline{\text{WE}}$ $\overline{\text{AS}}$	Negated
	$\overline{\text{SDMA}}[11-0]$ $\overline{\text{SDMA12/SDBA1}}$ $\overline{\text{SDBA0}}$	High impedance
Clock	$\text{PCI_CLK}[0-4]$ PCI_SYNC_OUT $\text{SDRAM_CLK}[0-3]$ SDRAM_SYNC_OUT CKO	Driven
Test/Configuration	$\overline{\text{TDO}}$	Negated

3.2 Detailed Signal Descriptions

The following subsections describe the MPC8240 input and output signals, and the meaning of their different states and relative timing information for assertion and negation. In cases where signals serve multiple functions (and have multiple names), they are described individually for each function.

3.2.1 PCI Interface Signals

This section provides descriptions of the PCI interface signals on the MPC8240. Note that throughout this manual, signals and bits of the PCI interface are referenced in little-endian

format. For more information on the operation of the MPC8240 PCI interface, see Chapter 8, “PCI Bus Interface.” Refer to the PCI Local Bus Specification, Revision 2.1, and PCI System Design Guide, Revision 1 for a thorough description of the PCI local bus and specific signal-to-signal timing relationships for the PCI bus.

3.2.1.1 PCI Address/Data Bus (AD[31–0])

The PCI address/data bus (AD[31–0]) consists of 32 signals that are both input and output signals on the MPC8240.

3.2.1.1.1 Address/Data (AD[31–0])—Output

Following is the state meaning for AD[31–0] as outputs.

State Meaning Asserted/Negated—Represents the physical address during the address phase of a PCI transaction. During a data phase of a PCI transaction, AD[31–0] contain data being written.

The AD[7–0] signals define the least-significant byte and AD[31–24] the most-significant byte.

3.2.1.1.2 Address/Data (AD[31–0])—Input

Following is the state meaning for AD[31–0] as inputs.

State Meaning Asserted/Negated—Represents the address to be decoded as a check for device select during an address phase of a PCI transaction or data being received during a data phase of a PCI transaction.

3.2.1.2 Command/Byte Enable ($\overline{C/BE}$ [3–0])

The four command/byte enable ($\overline{C/BE}$ [3–0]) signals are both input and output signals on the MPC8240.

3.2.1.2.1 Command/Byte Enable ($\overline{C/BE}$ [3–0])—Output

Following is the state meaning for $\overline{C/BE}$ [3–0] as output signals.

State Meaning Asserted/Negated—During the address phase, $\overline{C/BE}$ [3–0] define the bus command of the transaction initiated by the MPC8240 as a PCI master. Table 3-3 summarizes the PCI bus command encodings. See Section 8.3.2, “PCI Bus Commands,” for more detailed information on the bus commands.

During the data phase, $\overline{C/BE}$ [3–0] are used as byte enables. Byte enables determine which byte lanes carry meaningful data. The $\overline{C/BE0}$ signal applies to the least-significant byte.

Table 3-3. PCI Command Encodings

$\overline{C/BE}$ [3–0]	PCI Command
0000	Interrupt acknowledge
0001	Special cycle
0010	I/O read

Table 3-3. PCI Command Encodings (Continued)

$\overline{C/BE}[3-0]$	PCI Command
0011	I/O write
0100	Reserved
0101	Reserved
0110	Memory read
0111	Memory write
1000	Reserved
1001	Reserved
1010	Configuration read ¹
1011	Configuration write ¹
1100	Memory read multiple
1101	Double (dual) access cycle ²
1110	Memory read line
1111	Memory write and invalidate

¹ The MPC8240 does not respond to this command.

² The MPC8240 does not generate this command.

3.2.1.2.2 Command/Byte Enable ($\overline{C/BE}[3-0]$)—Input

Following is the state meaning for $\overline{C/BE}[3-0]$ as input signals.

State Meaning Asserted/Negated—During the address phase, $\overline{C/BE}[3-0]$ indicate the command that another master is sending. The MPC8240 uses the value on these signals (in addition to the address) to determine whether it is a target for a transaction. Table 3-3 summarizes the PCI bus command encodings. See Section 8.3.2, “PCI Bus Commands,” for more information.

During the data phase, $\overline{C/BE}[3-0]$ indicate which byte lanes are valid.

3.2.1.3 Device Select (\overline{DEVSEL})

The device select (\overline{DEVSEL}) signal is both an input and output on the MPC8240.

3.2.1.3.1 Device Select (\overline{DEVSEL})—Output

Following is the state meaning for \overline{DEVSEL} as an output.

State Meaning Asserted—Indicates that the MPC8240 has decoded the address of a PCI transaction, and it is the target of the current access.

Negated—Indicates that the MPC8240 has decoded the address and is not the target of the current access.

3.2.1.3.2 Device Select ($\overline{\text{DEVSEL}}$)—Input

Following is the state meaning for $\overline{\text{DEVSEL}}$ as an input signal.

State Meaning	<p>Asserted—Indicates that some PCI target (other than the MPC8240) has decoded its address as the target of the current access. This is useful to the MPC8240 when it is the initiator of a PCI transaction.</p> <p>Negated—Indicates that no PCI target has been selected.</p>
----------------------	--

3.2.1.4 Frame ($\overline{\text{FRAME}}$)

The frame ($\overline{\text{FRAME}}$) signal is both an input and output on the MPC8240.

3.2.1.4.1 Frame ($\overline{\text{FRAME}}$)—Output

Following is the state meaning for $\overline{\text{FRAME}}$ as an output.

State Meaning	<p>Asserted—Indicates that the MPC8240, acting as a PCI master, is initiating a bus transaction. While $\overline{\text{FRAME}}$ is asserted, data transfers may continue.</p> <p>Negated—If $\overline{\text{IRDY}}$ is asserted, indicates that the PCI transaction is in the final data phase. If $\overline{\text{IRDY}}$ is negated, it indicates that the PCI bus is idle.</p>
----------------------	---

3.2.1.4.2 Frame ($\overline{\text{FRAME}}$)—Input

Following is the state meaning for $\overline{\text{FRAME}}$ as an input signal.

State Meaning	<p>Asserted—Indicates that another PCI master is initiating a bus transaction and causes the MPC8240 to decode the address and the command signals to see if it is the target of the transaction.</p> <p>Negated—Indicates that the transaction is in the final data phase or that the bus is idle.</p>
----------------------	---

3.2.1.5 PCI Bus Grant ($\overline{\text{GNT}}[4-0]$)—Output

The PCI bus grant ($\overline{\text{GNT}}[4-0]$) signals are outputs on the MPC8240 and they have a different meaning depending on whether the MPC8240 PCI arbiter is enabled or disabled. The PCI $\overline{\text{GNT}}$ signals are point-to-point; every master has its own $\overline{\text{GNT}}$ signal.

Note that $\overline{\text{GNT}}4$ serves as a reset configuration signal that enables the debug address signals.

3.2.1.5.1 PCI Bus Grant ($\overline{\text{GNT}}[4-0]$)—Internal Arbiter Enabled

The MPC8240 PCI arbiter is enabled by a low value on the reset configuration pin MAA2 or by the setting of bit 15 of the PCI arbitration control register. In this case, the $\overline{\text{GNT}}[4-0]$ signals are used in conjunction with the $\overline{\text{REQ}}[4-0]$ signals as the arbiter for up to five PCI masters. Following is the state meaning for the $\overline{\text{GNT}}[4-0]$ input signals in this case.

State Meaning Asserted—The MPC8240 has granted control of the PCI bus to a requesting master, using the priority scheme described in Section 8.2, “PCI Bus Arbitration”. The MPC8240 will assert only one $\overline{\text{GNT}}$ signal during any clock cycle.

Negated—Indicates that the MPC8240 has not granted control of the PCI bus and external devices may not initiate a PCI transaction.

3.2.1.5.2 PCI Bus Grant ($\overline{\text{GNT}}[4-0]$)—Internal Arbiter Disabled

The MPC8240 PCI arbiter is disabled by a high value on the reset configuration pin MAA2 or by the clearing of bit 15 of the PCI arbitration control register. In this case, the $\overline{\text{GNT}}_0$ becomes the PCI bus request output for the MPC8240, and it is asserted when the MPC8240 needs to run a PCI transaction. If the $\overline{\text{REQ}}_0$ input signal is asserted prior to the need to run a PCI transaction, then the $\overline{\text{GNT}}_0$ signal will not assert (the bus is parked) when a PCI transaction is to be run. Following is the state meaning for the $\overline{\text{GNT}}[4-0]$ input signals when the internal arbiter is disabled.

State Meaning Asserted—The MPC8240 asserts the $\overline{\text{GNT}}_0$ signal as the PCI bus request output signal. $\overline{\text{GNT}}[4-1]$ signals do not assert in this case.

Negated—The $\overline{\text{GNT}}[4-1]$ signals are driven high (negated) in this mode. $\overline{\text{GNT}}_0$ is negated when the MPC8240 is not requesting control of the PCI bus or the bus is parked on the MPC8240.

3.2.1.6 PCI Bus Request ($\overline{\text{REQ}}[4-0]$)—Input

The PCI bus request ($\overline{\text{REQ}}[4-0]$) signals are inputs on the MPC8240, and they have a different meaning depending on whether the MPC8240 PCI arbiter is enabled or disabled. The PCI $\overline{\text{REQ}}$ signals are point-to-point, and every master has its own $\overline{\text{REQ}}$ signal.

3.2.1.6.1 PCI Bus Request ($\overline{\text{REQ}}[4-0]$)—Internal Arbiter Enabled

The MPC8240 PCI arbiter is enabled by a low value on the reset configuration pin MAA2 or by the setting of bit 15 of the PCI arbitration control register. In this case, the $\overline{\text{REQ}}[4-0]$ signals are used in conjunction with the $\overline{\text{GNT}}[4-0]$ signals as the arbiter for up to five PCI masters. Following is the state meaning for the $\overline{\text{REQ}}[4-0]$ input signals in this case.

State Meaning Asserted—External devices are requesting control of the PCI bus. The MPC8240 acts on the requests as described in Section 8.2, “PCI Bus Arbitration.”

Negated—Indicates that no external devices are requesting the use of the PCI bus.

3.2.1.6.2 PCI Bus Request ($\overline{\text{REQ}}[4-0]$)—Internal Arbiter Disabled

The MPC8240 PCI arbiter is disabled by a high value on the reset configuration pin MAA2 or by the clearing of bit 15 of the PCI arbitration control register. In this case, the $\overline{\text{REQ}}_0$ becomes the PCI bus grant input for the MPC8240, and it is asserted when the external arbiter is granting the use of the PCI bus to the MPC8240. Note that if the $\overline{\text{REQ}}_0$ input signal is asserted prior to the need to run a PCI transaction, then the MPC8240 $\overline{\text{GNT}}_0$ signal will not assert (the bus is parked) when a PCI transaction is to be run.

The $\overline{\text{REQ}}[4-1]$ input signals are ignored when the internal arbiter is disabled. Following is the state meaning of the $\overline{\text{REQ}}0$ signal when the internal arbiter is disabled.

State Meaning Asserted—The $\overline{\text{REQ}}0$ signal indicates that the MPC8240 is granted control of the PCI bus. If $\overline{\text{REQ}}0$ is asserted before the MPC8240 has a transaction to perform (that is, the MPC8240 is parked), the MPC8240 drives AD[31-0], $\overline{\text{C}}/\overline{\text{BE}}[3-0]$, and PAR to stable (but meaningless) states until they are needed for a legitimate transaction.

Negated— $\overline{\text{REQ}}0$ is negated when the MPC8240 is not granted control of the PCI bus.

3.2.1.7 Initiator Ready ($\overline{\text{IRDY}}$)

The initiator ready ($\overline{\text{IRDY}}$) signal is both an input and output on the MPC8240.

3.2.1.7.1 Initiator Ready ($\overline{\text{IRDY}}$)—Output

Following is the state meaning for $\overline{\text{IRDY}}$ as an output.

State Meaning Asserted—Indicates that the MPC8240, acting as a PCI master, can complete the current data phase of a PCI transaction. During a write, the MPC8240 asserts $\overline{\text{IRDY}}$ to indicate that valid data is present on AD[31-0]. During a read, the MPC8240 asserts $\overline{\text{IRDY}}$ to indicate that it is prepared to accept data.

Negated—Indicates that the PCI target needs to wait before the MPC8240, acting as a PCI master, can complete the current data phase. During a write, the MPC8240 negates $\overline{\text{IRDY}}$ to insert a wait cycle when it cannot provide valid data to the target. During a read, the MPC8240 negates $\overline{\text{IRDY}}$ to insert a wait cycle when it cannot accept data from the target.

3.2.1.7.2 Initiator Ready ($\overline{\text{IRDY}}$)—Input

Following is the state meaning for $\overline{\text{IRDY}}$ as an input signal.

State Meaning Asserted—Indicates another PCI master is able to complete the current data phase of a transaction.

Negated—If $\overline{\text{FRAME}}$ is asserted, it indicates a wait cycle from another master. This is used by the MPC8240 to insert wait cycles when it is a target of a PCI transaction. If $\overline{\text{FRAME}}$ is negated, it indicates the PCI bus is idle.

3.2.1.8 Lock ($\overline{\text{LOCK}}$)—Input

The lock ($\overline{\text{LOCK}}$) signal is an input on the MPC8240. See Section 8.5, “Exclusive Access,” for more information. Following is the state meaning for the $\overline{\text{LOCK}}$ input signal.

State Meaning Asserted—Indicates that a master is requesting exclusive access to memory, which may require multiple transactions to complete.

Negated—Indicates that a normal operation is occurring on the bus, or an access to a locked target is occurring.

3.2.1.9 Parity (PAR)

The PCI parity (PAR) signal is both an input and output signal on the MPC8240. See Section 8.6.1, “PCI Parity,” for more information on PCI parity.

3.2.1.9.1 Parity (PAR)—Output

Following is the state meaning for PAR as an output signal.

State Meaning Asserted—This signal is driven by the MPC8240 to indicate odd parity across the AD[31–0] and $\overline{C/BE}$ [3–0] signals (driven by the MPC8240) during the address and data phases of a transaction.

 Negated—Indicates even parity across the AD[31–0] and $\overline{C/BE}$ [3–0] signals driven by the MPC8240 during address and data phases.

3.2.1.9.2 Parity (PAR)—Input

Following is the state meaning for PAR as an input signal.

State Meaning Asserted—Indicates odd parity driven by another PCI master or the PCI target during read data phases.

 Negated—Indicates even parity driven by another PCI master or the PCI target during read data phases.

3.2.1.10 Parity Error (\overline{PERR})

The PCI parity error (\overline{PERR}) signal is both an input and output signal on the MPC8240. See Section 13.2.3.2, “Parity Error (PERR)” for more information on how the MPC8240 is set up to report parity errors.

3.2.1.10.1 Parity Error (\overline{PERR})—Output

Following is the state meaning for \overline{PERR} as an output signal.

State Meaning Asserted—Indicates that the MPC8240, acting as a PCI agent, detected a data parity error. (The PCI initiator drives \overline{PERR} on read operations; the PCI target drives \overline{PERR} on write operations.)

 Negated—Indicates no error.

3.2.1.10.2 Parity Error (\overline{PERR})—Input

Following is the state meaning for \overline{PERR} as an input signal.

State Meaning Asserted—Indicates that another PCI agent detected a data parity error while the MPC8240 was sourcing data (the MPC8240 was acting as the PCI initiator during a write, or was acting as the PCI target during a read).

 Negated—Indicates no error.

3.2.1.11 System Error (\overline{SERR})

The PCI system error (\overline{SERR}) signal is both an input and output signal on the MPC8240. It is an open-drain signal and can be driven by multiple devices on the PCI bus. Refer to

Section 13.2.3.1, “System Error (SERR),” for more information on how the MPC8240 drives and reports system errors.

3.2.1.11.1 System Error ($\overline{\text{SERR}}$)—Output

Following is the state meaning for $\overline{\text{SERR}}$ as an output signal.

State Meaning Asserted—Indicates that an address parity error, a target-abort (when the MPC8240 is acting as the initiator), or some other system error (where the result is a catastrophic error) was detected.

Negated—Indicates no error.

3.2.1.11.2 System Error ($\overline{\text{SERR}}$)—Input

Following is the state meaning for $\overline{\text{SERR}}$ as an input signal.

State Meaning Asserted—Indicates that a target (other than the MPC8240) has detected a catastrophic error.

Negated—Indicates no error.

3.2.1.12 Stop ($\overline{\text{STOP}}$)

The stop ($\overline{\text{STOP}}$) signal is both an input and output signal on the MPC8240. Refer to Section 8.4.3.2, “Target-Initiated Termination,” for more information on the use of the $\overline{\text{STOP}}$ signal.

3.2.1.12.1 Stop ($\overline{\text{STOP}}$)—Output

Following is the state meaning for $\overline{\text{STOP}}$ as an output signal.

State Meaning Asserted—Indicates that the MPC8240, acting as a PCI target, is requesting that the initiator stop the current transaction.

Negated—Indicates that the current transaction can continue.

3.2.1.12.2 Stop ($\overline{\text{STOP}}$)—Input

Following is the state meaning for $\overline{\text{STOP}}$ as an input signal.

State Meaning Asserted—Indicates that when the MPC8240 is acting as a PCI initiator, it is receiving a request from the target to stop the current transaction.

Negated—Indicates that the current transaction can continue.

3.2.1.13 Target Ready ($\overline{\text{TRDY}}$)

The target ready ($\overline{\text{TRDY}}$) signal is both an input and output signal on the MPC8240.

3.2.1.13.1 Target Ready ($\overline{\text{TRDY}}$)—Output

Following is the state meaning for $\overline{\text{TRDY}}$ as an output signal.

State Meaning Asserted—Indicates that the MPC8240, acting as a PCI target, can complete the current data phase of a PCI transaction. During a read, the MPC8240 asserts $\overline{\text{TRDY}}$ to indicate that valid data is present on AD[31–0]. During a write, the MPC8240 asserts $\overline{\text{TRDY}}$ to indicate that it is prepared to accept data.

Negated—Indicates that the PCI initiator needs to wait before the MPC8240, acting as a PCI target, can complete the current data phase. During a read, the MPC8240 negates $\overline{\text{TRDY}}$ to insert a wait cycle when it cannot provide valid data to the initiator. During a write, the MPC8240 negates $\overline{\text{TRDY}}$ to insert a wait cycle when it cannot accept data from the initiator.

3.2.1.13.2 Target Ready ($\overline{\text{TRDY}}$)—Input

Following is the state meaning for $\overline{\text{TRDY}}$ as an input signal.

State Meaning Asserted—Indicates another PCI target is able to complete the current data phase of a transaction. If the MPC8240 is the initiator of the transaction, it latches the data (on a read) or cycles the data on a write.

Negated—Indicates a wait cycle needed by a target. If the MPC8240 is the initiator of the transaction, it waits to latch the data (on a read) or continues to drive the data (on a write).

3.2.1.14 Interrupt Request ($\overline{\text{INTA}}$)—Output

Following is the state meaning for $\overline{\text{INTA}}$. This signal is only used when the MPC8240 is programmed in agent mode.

State Meaning Asserted—Indicates that the MPC8240 is requesting an interrupt on the PCI bus. These interrupts are caused by the on-chip DMA controller and the I₂O message unit.

Negated—Indicates that the MPC8240 is not requesting an interrupt on the PCI bus.

3.2.1.15 ID Select (IDSEL)—Input

Following is the state meaning for IDSEL. See Section 8.4.5.2, “Accessing the PCI Configuration Space,” for more information about the role of the IDSEL signal in PCI configuration transactions.

State Meaning Asserted—When the $\overline{\text{C/BE}}[3-0]$ encoding is set to configuration read/write, IDSEL indicates that the PCI configuration registers on the MPC8240 are being accessed.

Negated—Indicates that there is no configuration access for this device in progress.

Note that the MPC8240 must not issue PCI configuration transactions to itself (that is, for PCI configuration transactions initiated by the MPC8240, its IDSEL signal must not be asserted). The MPC8240 must use the method described in Section 5.1, “Configuration Register Access,” to access its own configuration registers. If the MPC8240 is in host mode and other PCI agents do not need to access the MPC8240’s configuration space, then it is recommended that this signal be pulled down.

3.2.2 Memory Interface Signals

The memory interface supports either standard DRAMs, extended data out DRAMs (EDO DRAMs), or synchronous DRAMs (SDRAMs) and either standard ROM or Flash devices. Some of the memory interface signals perform different functions (and are described by an alternate name) depending on the RAM and ROM configurations. This section provides a brief description of the memory interface signals on the MPC8240, listed individually by both their primary and alternate names, describing the relevant function in each section. For more information on the operation of the memory interface, see Chapter 6, “MPC8240 Memory Interface.”

3.2.2.1 Row Address Strobe ($\overline{\text{RAS}}[0-7]$)—Output

The eight row address strobe ($\overline{\text{RAS}}[0-7]$) signals are outputs on the MPC8240. Following are the state meaning and timing comments for the $\overline{\text{RAS}}_n$ output signals.

- State Meaning** Asserted—Indicates that the memory row address is valid and selects one of the rows in the selected bank for DRAM memory.
 Negated—Indicates DRAM precharge period.
- Timing Comments** Assertion—The MPC8240 asserts the $\overline{\text{RAS}}_n$ signal to begin a memory cycle. All other memory interface signal timings are referenced to $\overline{\text{RAS}}_n$.

3.2.2.2 Column Address Strobe ($\overline{\text{CAS}}[0-7]$)—Output

The eight column address strobe ($\overline{\text{CAS}}[0-7]$) signals are outputs on the MPC8240. $\overline{\text{CAS}}_0$ connects to the most-significant byte select. $\overline{\text{CAS}}_7$ connects to the least-significant byte select. When the MPC8240 is operating in 32-bit mode (see MCCR1[DBUS_SIZ[0-1]), the $\overline{\text{CAS}}[0-3]$ signals are used. Following are the state meaning and timing comments for the $\overline{\text{CAS}}_n$ output signals.

- State Meaning** Asserted—Indicates that the DRAM (or EDO) column address is valid and selects one of the columns in the row.
 Negated—For DRAMs, it indicates $\overline{\text{CAS}}_n$ precharge, and the current DRAM data transfer has completed.
 —or—
 For EDO DRAMs, it indicates $\overline{\text{CAS}}_n$ precharge, and the current data transfer completes in the first clock cycle of $\overline{\text{CAS}}_n$ precharge.
- Timing Comments** Assertion—The MPC8240 asserts $\overline{\text{CAS}}_n$ two to eight clock cycles after the assertion of $\overline{\text{RAS}}_n$ (depending on the setting of the MCCR3[RCD₂] parameter). See Section 6.3.5, “FPM or EDO DRAM Interface Timing,” for more information.

3.2.2.3 SDRAM Command Select ($\overline{\text{CS}}[0-7]$)—Output

The eight SDRAM command select ($\overline{\text{CS}}[0-7]$) signals are output on the MPC8240. Following are the state meaning and timing comments for the $\overline{\text{CS}}_n$ output signals.

- State Meaning** Asserted—Selects an SDRAM bank to perform a memory operation.
Negated—Indicates no SDRAM action during the current cycle.
- Timing Comments** Assertion—The MPC8240 asserts the \overline{CS}_n signal to begin a memory cycle. For SDRAM, \overline{CS}_n is valid on the rising edge of the SDRAM_CLK[0–3] clock signals.

3.2.2.4 SDRAM Data Qualifier (DQM[0–7])—Output

The eight SDRAM data qualifier (DQM[0–7]) signals are outputs on the MPC8240. Following are the state meaning and timing comments for the DQM_n output signals. DQM₀ connects to the most significant byte select, and DQM₇ connects to the least significant byte select.

- State Meaning** Asserted—Prevents writing to SDRAM. Note that the DQM_n signals are active-high for SDRAM. DQM_n is part of the SDRAM command encoding. See Section 6.4, “SDRAM Interface Operation,” for more information.
Negated—Allows a read or write operation to SDRAM.
- Timing Comments** Assertion—For SDRAM, DQM_n is valid on the rising edge of the SDRAM_CLK[0–3] clock signals during read or write cycles.

3.2.2.5 Write Enable (\overline{WE})—Output

The write enable (\overline{WE}) signal is an output on the MPC8240. Following are the state meaning and timing comments for the \overline{WE} output signal.

- State Meaning** For SDRAM, \overline{WE} is part of the SDRAM command encoding. See Section 6.4, “SDRAM Interface Operation,” for more information.
—otherwise—
Asserted—Enables writing to DRAM, EDO, or Flash.
Negated—No DRAM, EDO, or Flash write operation is pending.
- Timing Comments** Assertion—For DRAM, the MPC8240 asserts \overline{WE} concurrent with the column address and prior to \overline{CAS}_n . For SDRAM, the MPC8240 asserts \overline{WE} concurrent with \overline{SDCAS} for write operations.

3.2.2.6 SDRAM Address (SDMA[11–0])—Output

The SDMA[11–0] signals carry 13 of the address bits for the memory interface. For (S)DRAMs, they correspond to the row and column address bits.

- State Meaning** Asserted/Negated—Contain different portions of the address depending on the size of memory in use, the type of memory in use (DRAM, SDRAM, ROM or Flash) and the phase of the transaction. See Section 6.4.2, “SDRAM Address Multiplexing”, for a complete description of the mapping of these signals in all cases.
- Timing Comments** Assertion—For DRAM, the row address is considered valid on the assertion of \overline{RAS}_n , and the column address is valid on the assertion

of $\overline{\text{CAS}}_n$. For SDRAM, the row address is valid on the rising edge of SDRAM_CLK[0–3] clock signals when $\overline{\text{CS}}_n$ is asserted and the column address is valid on the rising edge of SDRAM_CLK[0–3] when DQM_n is asserted. For ROM and Flash, the address is valid with the assertion of either $\overline{\text{RCS}}_0$ or $\overline{\text{RCS}}_1$.

3.2.2.7 SDRAM Address 12 (SDMA12)—Output

The SDMA12 signal is similar to SDMA[11–0] in that it corresponds to different row or column address bits, depending on the memory in use.

State Meaning Asserted/Negated—See Section 6.4.2, “SDRAM Address Multiplexing,” for a complete description of the mapping of this signals in all cases.

Timing Comments Assertion/Negation—The same as SDMA[11–0].

3.2.2.8 SDRAM Internal Bank Select 1 (SDBA1)—Output

The SDBA1 signal is similar to SDMA[11–0] in that it corresponds to different row or column address bits, depending on the memory in use.

State Meaning Asserted/Negated—See Section 6.4.2, “SDRAM Address Multiplexing,” for a complete description of the mapping of this signals in all cases.

Timing Comments Assertion/Negation—The same as SDMA[11–0].

3.2.2.9 SDRAM Internal Bank Select 0 (SDBA0)—Output

The SDBA0 signal is similar to SDMA[11–0], but it is only used for the SDRAM interface.

State Meaning Asserted/Negated—Selects the SDRAM internal bank (Low = bank A, High = bank B) 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.

Timing Comments Assertion/Negation—The row address is valid on the rising edge of SDRAM_CLK[0–3] clock signals when $\overline{\text{CS}}_n$ is asserted and the column address is valid on the rising edge of SDRAM_CLK[0–3] when DQM_n is asserted.

3.2.2.10 Data Bus (DH[0–31], DL[0–31])

The data bus (DH[0–31], DL[0–31]) consists of 64 signals that are both input and output signals on the MPC8240. The data bus is comprised of two halves—data bus high (DH[0–31]) and data bus low (DL[0–31]). When the MPC8240 is operating in 32-bit mode (see MCCR1[DBUS_SIZ[0–1]), DH[0–31] is used, and DL[0–31] can be left floating. Table 3-4 specifies the byte lane assignments for the data bus.

Table 3-4. Data Bus Byte Lane Assignments

Data Bus Signals	Byte Lane
DH[0–7]	0 (MSB)
DH[8–15]	1
DH[16–23]	2
DH[24–31]	3
DL[0–7]	4
DL[8–15]	5
DL[16–23]	6
DL[24–31]	7 (LSB)

3.2.2.10.1 Data Bus (DH[0–31], DL[0–31])—Output

Following are the state meaning and timing comments for the data bus as output signals.

State Meaning Asserted/Negated—Represents the value of data being driven by the MPC8240.

Timing Comments Assertion/Negation—For DRAM accesses, the data bus signals are valid when $\overline{\text{CAS}}[0-7]$ and $\overline{\text{WE}}$ are asserted. For SDRAM, the data bus signals are valid on the next rising edge of SDRAM_CLK[0–3] after DQM[0–7] is asserted for a write command. For Flash memory, the data bus signals are valid on the assertion of $\overline{\text{RCS0}}$.

3.2.2.10.2 Data Bus (DH[0–31], DL[0–31])—Input

Following are the state meaning and timing comments for the data bus as input signals. Note that DL0 is a reset configuration input signal.

State Meaning Asserted/Negated—Represents the value of data being driven by the memory subsystem on a read.

Timing Comments Assertion/Negation—For a memory read transaction, the data bus signals are valid at a time dependent on the memory interface configuration parameters. Refer to Chapter 5, “Configuration Registers,” and Chapter 6, “MPC8240 Memory Interface,” for more information.

3.2.2.11 Data Parity/ECC (PAR[0–7])

The eight data parity/ECC (PAR[0–7]) signals are both input and output signals on the MPC8240.

3.2.2.11.1 Data Parity (PAR[0–7])—Output

Following are the state meaning and timing comments for PAR[0–7] as output signals.

State Meaning Asserted/Negated—Represents the byte parity or ECC bits being written to memory (PAR0 is the most-significant parity bit and corresponds to byte lane 0 which is selected by $\overline{\text{CAS0}}$). The data parity signals are asserted or negated as appropriate to provide odd parity (including the parity bit) or ECC.

Timing Comments Assertion/Negation—PAR[0–7] are valid concurrent with DH[0–31] and DL[0–31].

3.2.2.11.2 Data Parity (PAR[0–7])—Input

Following are the state meaning and timing comments for PAR[0–7] as input signals.

State Meaning Asserted/Negated—Represents the byte parity or ECC bits being read from memory (PAR0 is the most-significant parity bit and corresponds to byte lane 0 which is selected by $\overline{\text{CAS0}}$).

Timing Comments Assertion/Negation—PAR[0–7] are valid concurrent with DH[0–31] and DL[0–31].

3.2.2.12 ROM Address 19–12 (AR[19–12])—Output

The ROM address 19–12 (AR[19–12]) signals are output signals only for the ROM address function. Note that these signals are both input and output signals for the memory parity function (PAR[0–7]). Following are the state meaning and timing comments for AR[19–12] as output signals.

State Meaning Asserted/Negated—Represents bits 19–12 of the ROM/Flash address. The other ROM address bits are provided by AR[0–10] as shown in Section 6.5.1, “ROM/Flash Address Multiplexing.”

Timing Comments Assertion/Negation—The ROM address is valid on assertion of RCS0 or RCS1.

3.2.2.13 SDRAM Clock Enable (CKE)—Output

The SDRAM clock enable (CKE) signal is an output on the MPC8240 (and it is also used as a reset configuration input signal). CKE is part of the SDRAM command encoding. See Section 6.4, “SDRAM Interface Operation,” for more information. Following are the state meaning and timing comments for the CKE output signal.

State Meaning Asserted—Enables the internal clock circuit of the SDRAM memory device.

Negated—Disables the internal clock circuit of the SDRAM memory device. Note that the MPC8240 negates CKE during certain system power-down situations.

Timing Comments Assertion—CKE is valid on the rising edge of the SDRAM_CLK[0–3] clock signals. See Section 6.4, “SDRAM Interface Operation,” for more information.

3.2.2.14 SDRAM Row Address Strobe ($\overline{\text{SDRAS}}$)—Output

The SDRAM row address strobe ($\overline{\text{SDRAS}}$) signal is an output on the MPC8240. Following are the state meaning and timing comments for the $\overline{\text{SDRAS}}$ output signal.

State Meaning Asserted/Negated— $\overline{\text{SDRAS}}$ is part of the SDRAM command encoding and is used for SDRAM bank selection during read or write operations. See Section 6.4, “SDRAM Interface Operation,” for more information.

Timing Comments Assertion— $\overline{\text{SDRAS}}$ is valid on the rising edge of the SDRAM clock when a $\overline{\text{CSn}}$ signal is asserted.

3.2.2.15 SDRAM Column Address Strobe ($\overline{\text{SDCAS}}$)—Output

The SDRAM column address strobe ($\overline{\text{SDCAS}}$) signal is an output on the MPC8240. Following are the state meaning and timing comments for the $\overline{\text{SDCAS}}$ output signal.

State Meaning Asserted— $\overline{\text{SDCAS}}$ is part of the SDRAM command encoding and is used for SDRAM column selection during read or write operations. See Section 6.4, “SDRAM Interface Operation,” for more information.

Negated— $\overline{\text{SDCAS}}$ is part of SDRAM command encoding used for SDRAM column selection during read or write operations.

Timing Comments Assertion—For SDRAM, $\overline{\text{SDCAS}}$ is valid on the rising edge of the SDRAM clock when a $\overline{\text{CSn}}$ signal is asserted.

3.2.2.16 ROM Bank 0 Select ($\overline{\text{RCS0}}$)—Output

The ROM bank 0 select ($\overline{\text{RCS0}}$) signal is an output on the MPC8240 (and a reset configuration input signal). Following are the state meaning and timing comments for the $\overline{\text{RCS0}}$ output signal.

State Meaning Asserted—Selects ROM bank 0 for a read access or Flash bank 0 for a read or write access.

Negated—Deselects bank 0, indicating no pending memory access to ROM/Flash.

Timing Comments Assertion—The MPC8240 asserts $\overline{\text{RCS0}}$ at the start of a ROM/Flash access cycle.

Negation—Controlled by the ROMFAL and ROMNAL parameters of the MCCR1 register.

3.2.2.17 ROM Bank 1 Select ($\overline{\text{RCS1}}$)—Output

The ROM bank 1 select ($\overline{\text{RCS1}}$) signal is an output on the MPC8240. Following are the state meaning and timing comments for the $\overline{\text{RCS1}}$ output signal.

State Meaning Asserted—Selects ROM bank 1 for a read access or Flash bank 1 for a read or write access.

Negated—Deselects bank 1, indicating no pending memory access to ROM/Flash.

Timing Comments Assertion—The MPC8240 asserts $\overline{RCS1}$ at the start of a ROM/Flash access cycle.

Negation—Controlled by the ROMFAL and ROMNAL parameters of the MCCR1 register.

3.2.2.18 Flash Output Enable (\overline{FOE})—Output

The Flash output enable (\overline{FOE}) signal is an output on the MPC8240 (and a reset configuration input signal). Following are the state meaning and timing comments for the \overline{FOE} output signal.

State Meaning Asserted—Enables Flash output for the current read access.

Negated—Indicates that there is currently no read access to Flash.

Note that the \overline{FOE} signal provides no indication of any write operation(s) to Flash.

Timing Comments Assertion—The MPC8240 asserts \overline{FOE} at the start of the Flash read cycle.

Negation—Controlled by the ROMFAL and ROMNAL parameters of the MCCR1 register.

3.2.2.19 Address Strobe (\overline{AS})—Output

The \overline{AS} output signal is used as a user-defined timing signal for the Port X interface. The assertion and pulse width are fully programmable with the ASFALL and ASRISE parameters in the MCCR2 register. \overline{AS} is also a reset configuration input signal.

State Meaning Asserted—Programmable number of clocks (ASFALL) from the assertion of $\overline{RCS0}$ or $\overline{RCS1}$

Negated—Programmable number of clocks (ASRISE) from the assertion of \overline{AS}

3.2.3 EPIC Control Signals

There are five EPIC interrupt control signals that have dual functions. The signals serve as five distinct incoming interrupt requests (IRQ[0–4]) when the EPIC unit is in discrete interrupt mode (defined by GCR[M] = 1 and EICR[SIE] = 0). When the EPIC unit is in the serial interrupt mode (GCR[M] = 1 and EICR[SIE] = 1) or pass-through mode (GCR[M] = 0), each signal takes on an alternate function. The protocol for the various modes of the EPIC unit are described in Chapter 12, “Embedded Programmable Interrupt Controller (EPIC).”

3.2.3.1 Discrete Interrupt 0–4 (IRQ[0-4])—Input

Following is the state meaning for the IRQ[0–4] signals (discrete interrupt mode). The polarity and sense of each of these signals is programmable. All of these inputs can be driven completely asynchronously. In pass-through mode, interrupts from external source IRQ0 are passed directly to the processor.

State Meaning Asserted/Negated—When the interrupt signal is asserted (according to the programmed polarity), the priority is checked by the EPIC unit, and the interrupt is conditionally passed to the processor as described in Chapter 12, “Embedded Programmable Interrupt Controller (EPIC).”

3.2.3.2 Serial Interrupt Mode Signals

The serial interrupt mode provides for up to 16 interrupts to be serially clocked in through the S_INT signal. The relative timing for these signals is described in Section 12.6.3, “Serial Interrupt Timing Protocol.”

3.2.3.2.1 Serial Interrupt Stream (S_INT)—Input

This signal represents the incoming interrupt stream in serial interrupt mode.

State Meaning Asserted/Negated—Represents the interrupts for up to 16 external interrupt sources with individually programmable sense and polarity. These interrupts are clocked in to the MPC8240 by the S_CLK signal.

3.2.3.2.2 Serial Interrupt Clock (S_CLK)—Output

This output serves as the serial clock that the external interrupt source must use for driving the 16 interrupts onto the S_INT signal.

State Meaning Asserted/Negated—The frequency of this clock signal is programmed in the serial interrupt configuration register.

3.2.3.2.3 Serial Interrupt Reset (S_RST)—Output

Following is the state meaning of the S_RST signal.

State Meaning Asserted/Negated—S_RST is asserted only once for two S_CLK cycles when the EPIC is programmed to the serial interrupt mode.

3.2.3.2.4 Serial Interrupt Frame ($\overline{\text{S_FRAME}}$)—Output

Following is the state meaning of the $\overline{\text{S_FRAME}}$ signal.

State Meaning Asserted/Negated—Synchronizes the serial interrupt sampling to interrupt source 00.

3.2.3.3 Local Interrupt ($\overline{\text{L_INT}}$)—Output

Following is the state meaning of the $\overline{\text{L_INT}}$ signal.

State Meaning Asserted/Negated—When the EPIC is programmed in pass-through mode, this output reflects the raw interrupts generated by the on-chip I₂O, I²C, and DMA controllers.

3.2.4 I²C Interface Control Signals

These two signals serve as a communication interconnect with other devices. All devices connected to these two signals must have open-drain or open-collector outputs. The logic

Detailed Signal Descriptions

AND function is performed on both of these signals with external pull-up resistors. Refer to the *MPC8240 Hardware Specification* for the electrical characteristics of these signals.

Chapter 11, “I²C Interface,” has a complete description of the I²C protocol and the relative timings of the I²C signals.

3.2.4.1 Serial Data (SDA)

This signal is an input when the MPC8240 is in a receiving mode and an output when it is transmitting (as an I²C master or a slave).

3.2.4.1.1 Serial Data (SDA)—Output

Following is the state meaning of the SDA output signal when the MPC8240 is transmitting (as an I²C master or a slave).

State Meaning Asserted/Negated—Used to drive the data

3.2.4.1.2 Serial Data (SDA)—Input

Following is the state meaning of the SDA input signal when the MPC8240 is receiving data.

State Meaning Asserted/Negated—Used to receive data from other devices. The bus is assumed to be busy when SDA is detected low.

3.2.4.2 Serial Clock (SCL)

This signal is an input when the MPC8240 is programmed as an I²C slave and an output when programmed as an I²C master.

3.2.4.2.1 Serial Clock (SCL)—Output

Following is the state meaning of the SCL output signal when the MPC8240 is an I²C master.

State Meaning Asserted/Negated—Driven along with SDA as the clock for the data.

3.2.4.2.2 Serial Clock (SCL)—Input

Following is the state meaning of the SCL output signal when the MPC8240 is an I²C slave.

State Meaning Asserted/Negated—The I²C unit uses this signal to synchronize incoming data on SCL. The bus is assumed to be busy when this signal is detected low.

3.2.5 System Control and Power Management Signals

The following sections describe the system control and power management signals of the MPC8240.

3.2.5.1 Hard Reset

The two hard reset signals on the MPC8240 ($\overline{\text{HRST_CPU}}$ and $\overline{\text{HRST_CTRL}}$) must be asserted and negated together to guarantee normal operation. Together, $\overline{\text{HRST_CPU}}$ and $\overline{\text{HRST_CTRL}}$ cause the MPC8240 to abort all current internal and external transactions,

and set all registers to their default values. Although $\overline{\text{HRST_CPU}}$ and $\overline{\text{HRST_CTRL}}$ must be asserted together, they may be asserted completely asynchronously with respect to all other signals. See Section 13.2.1, “System Reset” for a complete description of the reset functionality.

3.2.5.1.1 Hard Reset (Processor) ($\overline{\text{HRST_CPU}}$)—Input

Should be asserted with $\overline{\text{HRST_CTRL}}$ to perform a complete hard reset of the MPC8240.

State Meaning Asserted/Negated—See Section 3.1.2, “Output Signal States at Reset,” and Section 3.4, “Configuration Pins Sampled at Reset,” for more information on the interpretation of the other MPC8240 signals during reset.

Timing Comments Assertion/Negation—See the *MPC8240 Hardware Specification* for specific timing information of these signals and the reset configuration signals.

3.2.5.1.2 Hard Reset (Peripheral Logic) ($\overline{\text{HRST_CTRL}}$)—Input

Should be asserted with $\overline{\text{HRST_CPU}}$ to perform a complete hard reset of the MPC8240.

State Meaning Asserted/Negated—See Section 3.1.2, “Output Signal States at Reset”, and Section 3.4, “Configuration Pins Sampled at Reset,” for more information on the interpretation of the other MPC8240 signals during reset.

Timing Comments Assertion/Negation—See the *MPC8240 Hardware Specification* for specific timing information of these signals and the reset configuration signals.

3.2.5.2 Soft Reset ($\overline{\text{SRESET}}$)—Input

The assertion of the soft reset input signal causes the same actions as the assertion of the internal $\overline{\text{sreset}}$ signal by the EPIC unit. A soft reset is recoverable, provided that in attempting to reach a recoverable state, the processor does not encounter a machine check condition. This exception is third in priority, following a hard reset and machine check.

State Meaning Asserted/Negated—When $\overline{\text{SRESET}}$ is asserted, the processor core attempts to reach a recoverable state by allowing the next instruction to either complete or cause an exception, blocking the completion of subsequent instructions, and allowing the completed store queue to drain. Unlike a hard reset, no registers or latches are initialized and the instruction cache is disabled.

Timing Comments Assertion—May occur at any time, asynchronous to any clock.
Negation—Must be asserted for at least 2 *sys_logic_clk* cycles. After $\overline{\text{SRESET}}$ is negated, the processor vectors to the system reset vector.

3.2.5.3 Machine Check ($\overline{\text{MCP}}$)—Output

The MCP signal is driven by the MPC8240 when a machine check error is generated by any of the conditions described in Chapter 13, “Error Handling and Exceptions” for generating

Detailed Signal Descriptions

the internal \overline{mcp} signal. The assertion of \overline{mcp} depends upon whether the error handling registers of the MPC8240 are set to report the specific error. Additionally, the programmable parameter PICR1[MCP_EN] is used to enable or disable the assertion of \overline{mcp} by the MPC8240 for all error conditions. \overline{MCP} is also used as a reset configuration input signal.

State Meaning Asserted—Reflects the state of the internal \overline{mcp} signal. The current transaction may or may not be aborted depending upon the software configuration. Assertion of \overline{mcp} causes the processor core to conditionally take a machine check exception or enter the checkstop state based on the setting of the MSR[ME] bit in the processor core.
Negated—There is no \overline{mcp} being reported to the processor core.

Timing Comments Assertion— \overline{mcp} may be asserted to the processor core on any cycle, so the same timing applies to \overline{MCP} .

Negation—The MPC8240 holds \overline{mcp} asserted until the processor core has taken the exception. The MPC8240 decodes a machine check acknowledge cycle by detecting processor reads from the two possible machine check exception addresses at 0x0000_0200–0x0000_0207 and 0xFFFF0_0200–0xFFFF0_0207; and then negates \overline{mcp} . This timing also applies to \overline{MCP} .

High impedance—If the PMCR2[SHARED_MCP] bit is set, the \overline{MCP} signal is tri-stated when there is no error to report.

3.2.5.4 Nonmaskable Interrupt (NMI)—Input

The nonmaskable interrupt (NMI) signal is an input on the MPC8240. Following are the state meaning and timing comments for the NMI input signal. See Chapter 13, “Error Handling and Exceptions,” for more information.

State Meaning Asserted—Indicates that the MPC8240 should signal a machine check interrupt (\overline{mcp}) to the processor core.
Negated—No NMI reported.

Timing Comments Assertion—NMI may occur at any time, asynchronously.
Negation—Should not occur until after the interrupt is taken.

3.2.5.5 System Management Interrupt (\overline{SMI})—Input

Following are the state meaning and timing comments for \overline{SMI} .

State Meaning Asserted—The \overline{SMI} input signal is level-sensitive, and causes exception processing for a system management interrupt when \overline{SMI} is asserted and MSR[EE] is set.
Negated—Indicates that normal operation should proceed.

Timing Comments Assertion—May occur at any time and may be asserted asynchronously to the input clocks.
Negation—Should not occur until the interrupt is taken.

3.2.5.6 Checkstop In ($\overline{\text{CHKSTOP_IN}}$)—Input

Following are the state meaning and timing comments for the $\overline{\text{CHKSTOP_IN}}$ signal.

State Meaning Asserted—Indicates that the MPC8240 processor core must terminate operation by internally gating off all clocks, and releasing all processor-related outputs to the high-impedance state.

Negated—Indicates that normal operation should proceed.

Timing Comments Assertion—May occur at any time and may be asserted asynchronously to the input clocks.

Negation—Must remain asserted until the system has been reset with a hard reset.

3.2.5.7 Time Base Enable (TBEN)—Input

Following are the state meaning and timing comments for TBEN.

State Meaning Asserted—Indicates that the time base should continue clocking. This input is essentially a count enable control for the time base counter.

Negated—Indicates that the time base should stop clocking.

Timing Comments Assertion/Negation—May occur on any cycle.

3.2.5.8 Quiesce Acknowledge ($\overline{\text{QACK}}$)—Output

The quiesce acknowledge ($\overline{\text{QACK}}$) signal is an output on the MPC8240. It is also a reset configuration input signal. See Chapter 14, “Power Management,” for more information about the power management signals. Following are the state meaning and timing comments for the $\overline{\text{QACK}}$ output signal.

State Meaning Asserted—Indicates that the processor core and peripheral logic are in either nap or sleep mode.

Negated—Indicates that the processor core and peripheral logic are not in nap or sleep mode.

3.2.5.9 Debug Signals

The following sections describe the debug signals used by the MPC8240 in various debug modes. See Chapter 15, “Debug Features,” for more details and timing information on the debug signals.

3.2.5.9.1 Memory Address Attributes (MAA[0–2])—Output

The memory attribute signals are associated with the memory interface and provide information about the source of the memory operation being performed by the MPC8240. They are also reset configuration input signals.

State Meaning Asserted/Negated—These signals are encoded to provide more detailed information about a memory transaction. See Section 15.1.1, “Memory Address Attribute Signals (MAA[0–2]),” for a table showing these encodings.

Timing Comments Assertion/Negation—Section 15.1.2, “Memory Address Attribute Signal Timing,” refers to timing diagrams showing the relative timing of these signals and the rest of the memory interface.

3.2.5.9.2 PCI Address Attributes (PMAA[0–2])—Output

The memory attribute signals are associated with the PCI interface and provide information about the source of the PCI operation being performed by the MPC8240. They are also reset configuration input signals.

State Meaning Asserted/Negated—These signals are encoded to provide more detailed information about a PCI transaction. See Section 15.1.3, “PCI Address Attribute Signals,” for a table showing these encodings.

Timing Comments Assertion/Negation—Section 15.1.4, “PCI Address Attribute Signal Timing,” contains timing diagrams showing the relative timing of these signals and the rest of the PCI interface.

3.2.5.9.3 Debug Address (DA[0–15])—Output

When enabled, the debug address provides software disassemblers a simple way to reconstruct the 30-bit physical address for a memory bus transaction to DRAM and SDRAM, ROM, Flash, or PortX. Note that most of these signals are multiplexed with other signals (that may be inputs in their alternate function).

State Meaning Asserted/Negated—Section 15.2, “Memory Debug Address,” describes these signals in detail, and how they are mapped to different address bits, depending on the type of memory in use.

Timing Comments Assertion/Negation— For DRAM or SDRAM, these 16 debug address signals are sampled with the column address, and chip-selects. For ROM, Flash, and PortX devices, the debug address pins are sampled at the same time as the ROM address and can be used to recreate the 24-bit physical address in conjunction with ROM address.

3.2.5.9.4 Memory Interface Valid (\overline{MIV})—Input

The \overline{MIV} signal is intended to help reduce the number of bus cycles that logic analyzers must store in memory during a debug trace by signalling when address and data signals should be sampled.

State Meaning Asserted—The memory interface valid signal, \overline{MIV} , is asserted whenever FPM, EDO, SDRAM, Flash, or ROM addresses or data are present on the external memory bus.

Timing Comments Assertion/Negation—Section 15.3.1, “MIV Signal Timing” describes the relative timing of \overline{MIV} in detail.

3.2.6 Test and Configuration Signals

The MPC8240 has several signals that are sampled during reset to determine the configuration of the ROM, Flash, and dynamic memory, and the phase-locked loop clock mode. This section describes the signals sampled during reset, and their configuration. Weak pull-up or pull-down resistors should be used to avoid interference with normal signal operations.

To facilitate system testing, the MPC8240 provides a JTAG test access port (TAP) that complies with the IEEE 1149.1 boundary-scan specification. This section describes the JTAG test access port signals.

3.2.6.1 PLL Configuration (PLL_CFG[0–4])—Input

PLL_CFG[0–4] determine the clock frequency relationships of the PCI clock, the processor core frequency, and the *sys_logic_clk* signal (that determines the frequency of the memory interface clock). The multiplier factor determined by these signals on reset is stored in HID1[PLLRATIO]. However, system software cannot read the PLLRATIO value and associate it with a unique PLL_CFG[0–4] value. See Section 2.3.1.2.2, “Hardware Implementation-Dependent Register 1 (HID1),” for more information on HID1.

State Meaning Asserted—See the *MPC8240 Hardware Specification* for the supported settings.

Timing Comments Assertion—These signals are sampled at the negation of HRST_CPU and HRST_CTRL as part of the reset configuration signals. See Section 3.4, “Configuration Pins Sampled at Reset.”

3.2.6.2 JTAG Test Clock (TCK)—Input

The JTAG test clock (TCK) signal is an input on the MPC8240. Following is the state meaning for the TCK input signal.

State Meaning Asserted/Negated—This input should be driven by a free-running clock signal with a 30–70% duty cycle. Input signals to the test access port are clocked in on the rising edge of TCK. Changes to the test access port output signals occur on the falling edge of TCK. The test logic allows TCK to be stopped.

Note that this input contains an internal pull-up resistor to ensure that an unterminated input appears as a high signal level to the test logic.

3.2.6.3 JTAG Test Data Input (TDI)—Input

Following is the state meaning for the TDI input signal.

State Meaning Asserted/Negated—The value presented on this signal on the rising edge of TCK is clocked into the selected JTAG test instruction or data register.

Note that this input contains an internal pull-up resistor to ensure that an unterminated input appears as a high signal level to the test logic.

3.2.6.4 JTAG Test Data Output (TDO)—Output

Following is the state meaning for the TDO output signal.

State Meaning Asserted/Negated—The contents of the selected internal instruction or data register are shifted out onto this signal on the falling edge of TCK. The TDO signal will remain in a high-impedance state except when scanning of data is in progress.

3.2.6.5 JTAG Test Mode Select (TMS)—Input

The test mode select (TMS) signal is an input on the MPC8240. Following is the state meaning for the TMS input signal.

State Meaning Asserted/Negated—This signal is decoded by the internal JTAG TAP controller to distinguish the primary operation of the test support circuitry.

Note that this input contains an internal pull-up resistor to ensure that an unterminated input appears as a high signal level to the test logic.

3.2.6.6 JTAG Test Reset ($\overline{\text{TRST}}$)—Input

The test reset ($\overline{\text{TRST}}$) signal is an input on the MPC8240. Following is the state meaning for the $\overline{\text{TRST}}$ input signal.

State Meaning Asserted—This input causes asynchronous initialization of the internal JTAG test access port controller. Note that the signal must be asserted during power-up reset in order to properly initialize the JTAG test access port.

Negated—Indicates normal operation.

Note that this input contains an internal pull-up resistor to ensure that an unterminated input appears as a high signal level to the test logic.

3.2.7 Clock Signals

The MPC8240 coordinates clocking across the memory bus and the PCI bus. This section provides a brief description of the MPC8240 clock signals. See Section 3.3, “Clocking,” for more detailed information on the use of the MPC8240 clock signals.

3.2.7.1 System Clock Input (OSC_IN)—Input

Provides the input to the PCI clock fanout buffer. A clock can be connected to this input to provide multiple low-skew copies on the PCI_CLK[0–4] and PCI_SYNC_OUT signals. For systems that do use the fanout buffer feature, this signal should be pulled high.

3.2.7.2 PCI Clock (PCI_CLK[0–4])—Output

These signals provide multiple copies of OSC_IN as output signals when using the PCI clock fanout buffer feature. If these outputs are not needed, they can be individually disabled in the ODCR register to minimize power consumption.

3.2.7.3 PCI Clock Synchronize Out (PCI_SYNC_OUT)—Output

This output is an additional clock provided by the PCI clock fanout buffer. It is intended to be fed into the PCI_SYNC_IN signal to allow the internal clock subsystem to synchronize to the system PCI clocks.

3.2.7.4 PCI Feedback Clock (PCI_SYNC_IN)—Input

This signal provides the input to the peripheral logic PLL. The PLL multiplies up and synchronizes to this reference clock. The frequency of the PLL outputs is based on the PLL clock frequency configuration signal settings at reset. See the *MPC8240 Hardware Specification* for a complete listing of supported PLL_CFG[0–4] settings.

3.2.7.5 SDRAM Clock Outputs (SDRAM_CLK[0–3])—Output

The MPC8240 provides four low-skew copies of the SDRAM clock for use in small memory subsystems. This clock is synchronized to the on-chip logic using a DLL. If these outputs are not needed, they can be individually disabled in the ODCR register to minimize power consumption.

3.2.7.6 SDRAM Clock Synchronize Out (SDRAM_SYNC_OUT)—Output

SDRAM_SYNC_OUT is additional copy of the SDRAM clock provided to allow feedback into the DLL to allow proper compensation for the output and flight time delay of the clock path.

3.2.7.7 SDRAM Feedback Clock (SDRAM_SYNC_IN)—Input

The SDRAM_SYNC_OUT signal should be connected to the SDRAM_SYNC_IN signal to allow the on-chip DLL to synchronize and compensate for routing delays and the output buffer.

For systems that use an external PLL to provide the clock source to SDRAM, this signal can be pulled high unless the SDRAM clock-to-PCI clock ratio is non-integer (3:2 or 5:2). In that case, this signal is used to synchronize between the internal clock and the external SDRAM clock

3.2.7.8 Debug Clock (CKO)—Output

The debug clock (CKO) signal is an output on the MPC8240. The internal signal reflected on CKO is determined by either the HID0[ECLK,SBCLK] bits (if PMCR1[CKO_SEL] = 0), or the two-bit PMCR1[CKO_MODE] field (if PMCR1[CKO_SEL] = 1). Both of these options allow the CKO output driver to be disabled. See Section 2.3.1.2.1, “Hardware Implementation-Dependent Register 0 (HID0),” and Section 5.3.1, “Power Management Configuration Register 1 (PMCR1),” for more information.

Note that as described in Section 2.3.1.2.1, “Hardware Implementation-Dependent Register 0 (HID0),” *sys-logic-clk* is driven on CKO while $\overline{\text{HRST_CPU}}$ and $\overline{\text{HRST_CTRL}}$ are asserted.

Clocking

The signal on this output is derived from a variety of internal signals after passing through differing numbers of internal buffers. This signal is intended for use during system debug; it is not intended as a reference clock signal.

3.3 Clocking

The following sections describe the clocking on the MPC8240.

3.3.1 Clocking Method

The MPC8240 allows for multiple clock options to suit the needs of various system configurations. Internally, the MPC8240 uses a phase-locked loop (PLL) circuit to generate master clocks to the system logic and a PLL to generate the processor clock. The system logic PLL is synchronized to the PCI_SYNC_IN input signal.

Figure 3-2 shows a block diagram of the clocking signals in the MPC8240.

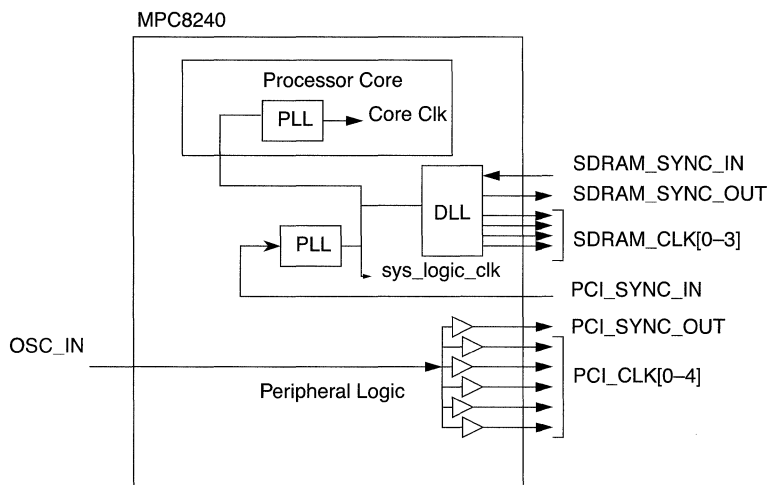


Figure 3-2. Clock Subsystem Block Diagram

The processor clock may be set to a multiple of the PCI bus frequency as defined in the *MPC8240 Hardware Specification*. To help reduce the amount of discrete logic required in a system, the MPC8240 provides PCI clock fan-out buffers as well as a Delay Locked Loop (DLL). The MPC8240 also provides the memory clock (SDRAM_CLK) signal through an additional DLL that is running at the same frequency as the internal system logic (*sys_logic_clk*).

Figure 3-3 shows the relationship of PCI_SYNC_IN and some multiplied clocks.

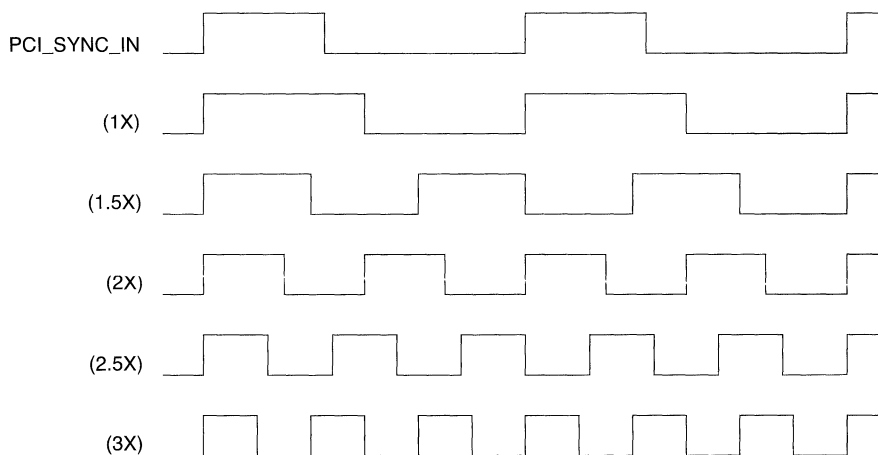


Figure 3-3. Timing Diagram (1X, 1.5X, 2X, 2.5X, and 3X examples)

Note

PCI_SYNC_IN is not required to have a 50% duty cycle. Furthermore, the bus interface clocks, internal clock and PCI_SYNC_IN edges are phase locked by the PLL.

The PLL is configured by the PLL_CFG[0–4] signals at reset. For a given PCI frequency, these signals set the peripheral logic frequency and PLL (VCO) frequency of operation, as well as determine the frequency for the processor core. The multiplier for the processor’s PLL is represented by the value in HID1[PLLRATIO]. See Section 2.3.1.2.2, “Hardware Implementation-Dependent Register 1 (HID1),” for more information on HID1. The supported settings for the PLL configuration pins are defined in the *MPC8240 Hardware Specification*.

3.3.2 DLL Operation and Locking

The DLL on the MPC8240 generates the SDRAM_CLK[0–3] and SDRAM_SYNC_OUT signals. SDRAM_SYNC_OUT should be fed back through a delay loop into the SDRAM_SYNC_IN input of the MPC8240. By adjusting the length of the delay loop, it is possible to remove the effects of trace delay to the system memory. This is accomplished when the delay through the loop is equivalent to the delay to the system memory.

There is a bit in the address map B options register (AMBOR) that controls the tap point of the DLL. See Section 5.8, “Address Map B Options Register,” for more information about the DLL_RESET bit.

The peripheral logic DLL is synchronized to SDRAM_SYNC_IN. Figure 3-2 shows how the PCI_SYNC_IN loop and SDRAM_SYNC_IN loop are independent of each other. These loops are supplied for synchronization of external components on the system board.

Clocking

The loop length on PCI_SYNC_IN should be designed so that it is the same (within 10% tolerance) as the loop lengths on the PCI_CLK signals to their driven components. Similarly, the loop length on and SDRAM_SYNC_IN should be designed so that it is the same (within 10% tolerance) as the loop lengths on the SDRAM_CLK signals to their driven components. For example, if a PCI device has a 5-inch trace, the loop trace must be 5 inches in length.

3.3.3 Clock Synchronization

The MPC8240 has the ability to provide the entire system with various system clocks based on PCI_SYNC_IN and the PLL_CFG[0–4] setting at reset. All of these clocks are synchronized by the internal logic of the MPC8240. In systems that use an external PLL to generate an external processor rate clock and do not depend on the SDRAM_CLK[0–3] signals, the MPC8240 can synchronize the internal processor bus to the external clock through the SDRAM_SYNC_IN signal.

In situations where the setting of PLL_CFG[0–4] creates a half-clock ratio between the PCI bus and processor bus, and an external PLL is being used to generate the external processor clock, then SDRAM_SYNC_IN must be driven by the external PLL the same way as the SDRAM devices. In addition, clock flipping logic must be enabled through the reset configuration pin \overline{QACK} . This flipping ensures that the internal processor bus rate will be synchronized to the external processor bus rate in half-clock modes. Also, by enabling the flipping during half-clock modes, the internal *hard_reset* to the processor core is delayed by 2^{17} (131072) processor clock cycles. This delay is required to insure that the clocking has been stabilized inside the MPC8240 after a reset.

3.3.4 Clocking System Solution Examples

This section describes two example clocking solutions for different system requirements. For systems where the MPC8240 is the host controller with a minimum number of clock loads, clock fanout buffers are provided on-chip (shown in Figure 3-4). For systems requiring more clock fanout or where the MPC8240 is an agent device, external clock buffers may be used as shown in Figure 3-5.

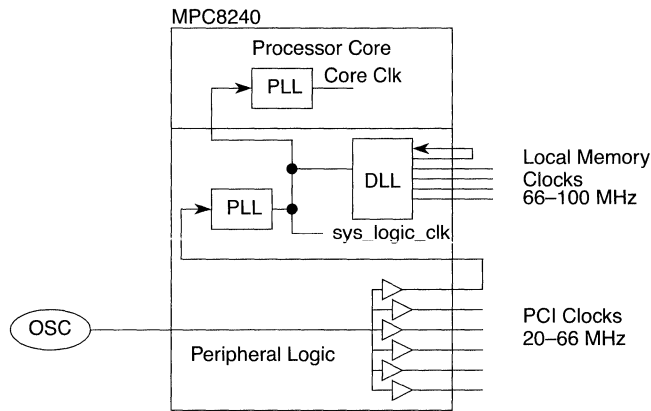


Figure 3-4. Clocking Solution—Small Load Requirements

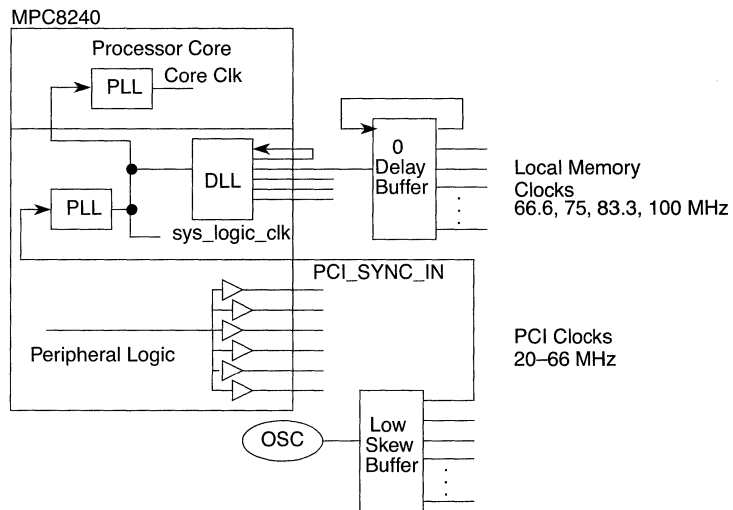


Figure 3-5. Clocking Solution—High Clock Fanout Required

3.4 Configuration Pins Sampled at Reset

Table 3-5 contains a description of the pins sampled for configuration at the negation of the HRST_CTRL and HRST_CPU signals. These configuration signals serve multiple purposes, and the signal names do not reflect the functionality of the signals as they are used for reset configuration. The values on these signals during reset are interpreted to be logic

one or zero, regardless of whether the signal name is defined as active-low. Most of the reset configuration signals have internal pull-up resistors so that if the signals are not driven, the default value is high (a one), as shown in the table. The PLL_CFG[4–0] signals do not have pull-up resistors and must be driven high or low during the reset period. For more information about the timing requirements of these configuration signals relative to the negation of the reset signals, refer to the *MPC8240 Hardware Specification*.

Table 3-5. Reset Configuration Signals

Signal Name(s)	Default	State Meaning
\overline{AS}	1	Clock out select—Sets the initial value of PMCR1[CKO_SEL]: 0 Processor CKO; value on this signal determined by HID0[ECLK,SBCLK]. 1 Peripheral logic CKO; value on this signal determined by PMCR1[CKO_MODE] field.
DL[0], \overline{FOE}	11	Sets the initial ROM bank 0 data path width, DBUS_SIZE[0–1], values in MCCR1. DBUS_SIZE[0–1] = (DL[0], \overline{FOE}) at reset. For ROM/FLASH chip select #0 ($\overline{RCS0}$), (DL[0] = 0, \overline{FOE} = 0) = 32-bit data bus. (DL[0] = x, \overline{FOE} = 1) = 8-bit data bus. (DL[0] = 1, \overline{FOE} = 0) = 64-bit data bus. For ROM/FLASH chip select #1 ($\overline{RCS1}$) and memory data bus, (DL[0] = 0, \overline{FOE} = x) = 32-bit data bus. (DL[0] = 1, \overline{FOE} = x) = 64-bit data bus.
MAA0	1	Initial address map—The setting of this signal during reset sets the initial ADDRESS_MAP value in the PICR1 register. 0 The MPC8240 is configured for address map A (not supported when operating in PCI agent mode). 1 The MPC8240 is configured for address map B.
MAA1	1	MPC8240 host mode 0 MPC8240 is a PCI agent device 1 MPC8240 is a PCI master (host) device
MAA2	1	PCI arbiter disable—The value on this signal is inverted and then written as the initial value of bit 15 in the PCI Arbiter Control Register. 0 PCI arbiter enabled 1 PCI arbiter disabled
MCP, CKE	11	PCI output hold delay value (in nanoseconds) relative to PCI_SYNC_IN. The values on these two signals determine the initial settings of PMCR2[6–5] as follows: 00 0.5 ns 01 1.3 ns 10 2.1 ns 11 2.9 ns (default if these pins not driven during reset) Note that the PMCR2 register has an additional bit (bit 4) that can be programmed to provide four more possible PCI output hold delay values. Refer to Section 5.3.2, “Power Management Configuration Register 2 (PMCR2)” for more information on these settings

Table 3-5. Reset Configuration Signals (Continued)

Signal Name(s)	Default	State Meaning
PMAA0	1	Driver capability for data bus (including \overline{QACK} and MAA[0–2] signals). Sets the initial value of the DRV_MEM_CTRL_1 bit in ODCR. Used in combination with PMAA1, as follows: 1 20- Ω data bus drive capability; when this is selected, only 8- Ω or 13.3- Ω drive capability allowed for PMAA1 0 40- Ω data bus drive capability; when this is selected, only 20- Ω or 40- Ω drive capability allowed for PMAA1
PMAA1	1	Driver capability for address signals (PAR[0–7], \overline{RAS} [0–7], CAS[0–7], \overline{WE} , \overline{FOE} , RCS0, RCS1, SDMA12, SDBA0, AR[19–12], SDRAS, SDCAS, CKE, AS, and SDMA[11–0]). Sets the initial value of the DRV_MEM_CTRL_2 bit in the ODCR register. The meaning of this signal setting depends on the setting of PMAA0. The two signals and the meaning of their combined settings for the address signals are shown below: [PMAA0, PMAA1]: 11 8- Ω drive capability 10 13.3- Ω drive capability 01 20- Ω drive capability 00 40- Ω drive capability
PMAA2	1	Driver capability for PCI and EPIC controller output signals. The value of this signal sets the initial value of ODCR[DRV_PCI]. 0 High drive capability on PCI signals (25 Ω) 1 Medium drive capability on PCI signals (50 Ω)
\overline{QACK}	1	Clock flip disable. When this signal is low on reset, it enables internal clock flipping logic, which is necessary when the PLL[0–4] signals select a half-clock frequency ratio. See Section 3.3.3, “Clock Synchronization” for more information on the use of clock flipping. 0 Clock flip enabled 1 No clock flip
RCS0	1	Boot memory location. The setting of this signal during reset sets the initial RCS0 value in the PICR1 register 0 Indicates that boot ROM is located on the PCI bus. 1 Indicates that boot ROM is located on local processor/memory data bus.
PLL_CFG[0–4]	must be driven	These five signals select the clock frequency ratios used by the two PLLs of the MPC8240. The value of PLL_CFG[0–4] during reset affects the read-only PLLRATIO field stored in HID1. Note that system software cannot associate the PLLRATIO value with a unique PLL_CFG[0–4] value. The MPC8240 Hardware Specification lists the supported settings and provides more detailed information on the clock frequencies.
$\overline{GNT4}$	1	Debug address enable. See Section 15.2, “Memory Debug Address”, for more information about this function. 0 Debug address enabled; partial address of the transaction driven on DA[0–15]. 1 Debug address disabled

Configuration Pins Sampled at Reset

Chapter 4

Address Maps

The MPC8240 in PCI host mode supports two address mapping configurations designated as address map A, and address map B. Address map A conforms to the PowerPC reference platform (PReP) specification. Address map B conforms to the PowerPC microprocessor common hardware reference platform (CHRP). When the MPC8240 is configured as a PCI agent, only map B is supported.

The memory map is selected at reset by a configuration pin. If the MPC8240 is programmed to host mode and the configuration pin is pulled low, the MPC8240 uses address map A. If the MPC8240 is programmed for host mode and the configuration pin is pulled high, the MPC8240 uses address map B. If the MPC8240 is programmed to agent mode, it also uses map B.

In addition, MPC8240 offers address translation capability to allow address remapping for inbound and outbound PCI memory transactions. Translation is necessary when MPC8240 is used as a PCI agent device. It allows a certain address space to map to a window of physical memory.

Note that the support of map A is provided for backward compatibility only. It is strongly recommended that new designs use map B because map A may not be supported in future devices.

4.1 Address Map A

Address map A complies with the PowerPC reference platform specification. The address space of map A is divided into four areas—system memory, PCI I/O, PCI memory, and system ROM space. Table 4-1, Table 4-2, and Table 4-3 show separate views of address map A for the processor core, a PCI memory device, and a PCI I/O device, respectively. When configured for map A, the MPC8240 translates addresses across the internal peripheral logic bus and the external PCI bus as shown in Figure 4-1 through Figure 4-3.

Table 4-1. Address Map A—Processor View

Processor Core Address Range				PCI Address Range	Definition
Hex		Decimal			
0000_0000	7FFF_FFFF	0	2G – 1	No PCI cycle	Local memory space
8000_0000	807F_FFFF	2G	2G + 8M – 1	0000_0000–007F_FFFF	PCI/ISA I/O space ^{1, 2}
8080_0000	80FF_FFFF	2G + 8M	2G + 16M – 1	0080_0000–00FF_FFFF	PCI configuration direct access ³
8100_0000	BF7F_FFFF	2G + 16M	3G – 8M – 1	0100_0000–3F7F_FFFF	PCI I/O space
BF80_0000	BFFF_FFEF	3G – 8M	3G – 16 – 1	3F80_0000–3FFF_FFEF	Reserved
BFFF_FFF0	BFFF_FFFF	3G – 16	3G – 1	3FFF_FFF0–3FFF_FFFF	PCI/ISA interrupt acknowledge
C000_0000	FEFF_FFFF	3G	4G – 16M – 1	0000_0000–3EFF_FFFF	PCI memory space
FF00_0000	FFFF_FFFF	4G – 16M	4G – 1	No PCI cycle ⁴	ROM space ⁴

Table 4-2. Address Map A—PCI Memory Master View

PCI Memory Transactions Address Range				Processor Core Address Range	Definition
Hex		Decimal			
0000_0000	00FF_FFFF	0	16M – 1	No local memory cycle	PCI/ISA memory space
0100_0000	7EFF_FFFF	16M	2G – 16M – 1	No local memory cycle	PCI memory space
7F00_0000	7FFF_FFFF	2G – 16M	2G – 1	No local memory cycle	Reserved
8000_0000	FFFF_FFFF	2G	4G – 1	0000_0000–7FFF_FFFF	System memory space

Table 4-3. Address Map A—PCI I/O Master View

PCI I/O Transactions Address Range				Processor Core Address Range	Definition
Hex		Decimal			
0000_0000	0000_FFFF	0	64K – 1	No local memory cycle	ISA/PCI I/O space
0001_0000	007F_FFFF	64K	8M – 1	No local memory cycle	Reserved
0080_0000	3F7F_FFFF	8M	1G – 8M – 1	No local memory cycle	PCI I/O space
3F80_0000	3FFF_FFFF	1G – 8M	1G – 1	No local memory cycle	Reserved
4000_0000	FFFF_FFFF	1G	4G – 1	No local memory cycle	Reserved

Notes:

1. PCI configuration accesses to CF8 and CFC–CFF are handled as specified in the PCI Local Bus Specification. See Section 8.4.5.2, “Accessing the PCI Configuration Space,” for more information on how the MPC8240 accesses PCI configuration space.
2. Processor addresses are translated to PCI addresses as follows:
 PCI address (AD[31–0]) = 0b0 || A[1–31]. PCI configuration accesses use processor addresses 8000_0CF8 and 8000_0CFC–8000_0CFF. Note only 64 Kbytes (8000_0000-8000_FFFF) has been defined while the rest are reserved for future use.
3. IDSEL for direct-access method: 11=0x8080_08xx, 12=0x8080_10xx, ..., 18=0x8084_00xx. See Section 5.1.1.2, “Direct Access Method.”
4. If the ROM is not local, these addresses are not reserved and PCI cycles are generated.

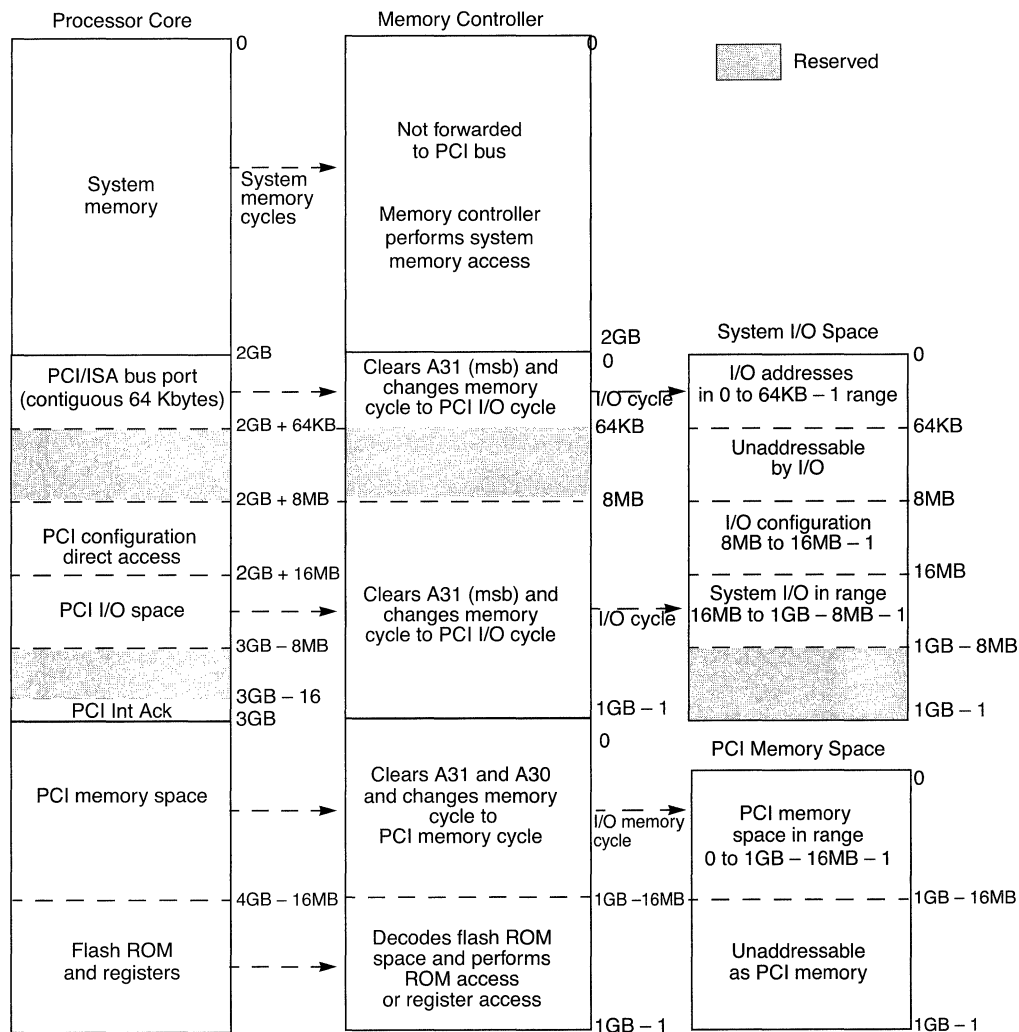


Figure 4-1. Processor Core Address Map A

Address Map A

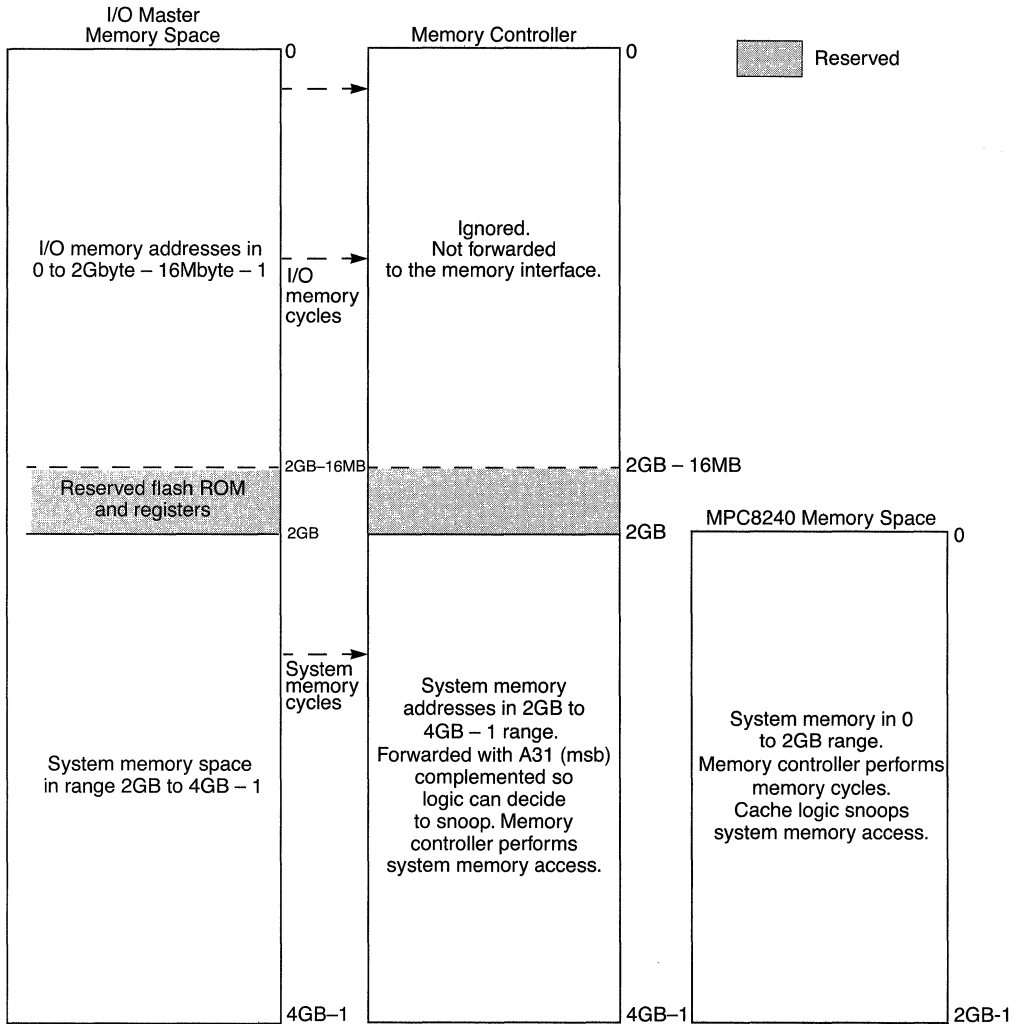


Figure 4-2. PCI Master Memory Address Map A

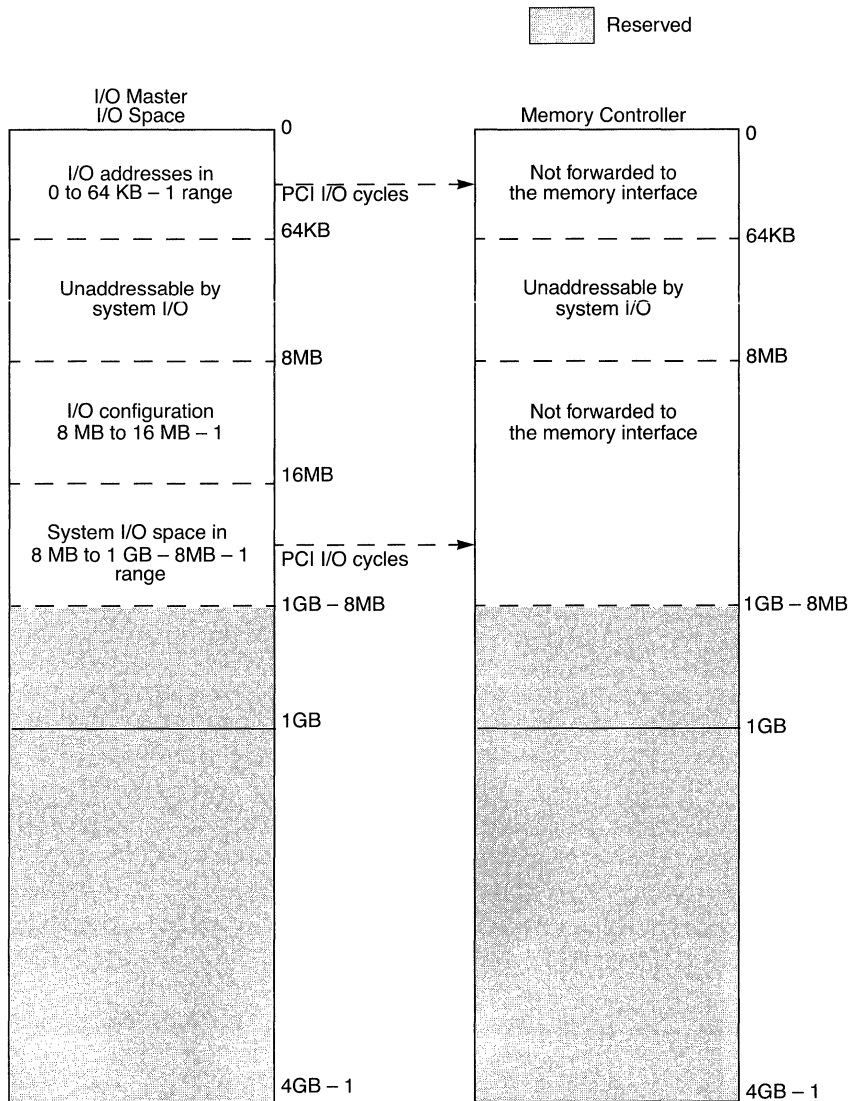


Figure 4-3. PCI Master I/O Address Map A

4.2 Address Map B

Address map B complies with the PowerPC microprocessor common hardware reference platform (CHRP). As with address map A, the address space of map B is divided into four areas—system memory, PCI memory, PCI I/O, and system ROM space. When configured for map B, the MPC8240 translates addresses across the internal peripheral logic bus and the external PCI bus as shown in Figure 4-4. Table 4-5, Table 4-5, and Table 4-7 show separate views of address map B for the processor core, a PCI memory device, and a PCI I/O device, respectively.

Address Map B

Clearing bit 16 of the processor interface configuration register (PICR) selects map B. MPC8240 only responds to addresses in system memory from the PCI perspective.

Table 4-4. Address Map B—Processor View

Processor Core Address Range				PCI Address Range	Definition
Hex		Decimal			
0000_0000	0009_FFFF	0	640K – 1	No PCI cycle	System memory space
000A_0000	000F_FFFF	640K	1M – 1	000A_0000–000F_FFFF	Compatibility hole ¹
0010_0000	3FFF_FFFF	1M	1G – 1	No PCI cycle	System memory space
4000_0000	7FFF_FFFF	1G	2G – 1	No PCI cycle	Reserved ²
8000_0000	FCFF_FFFF	2G	4G – 48M – 1	8000_0000–FCFF_FFFF	PCI memory space
FD00_0000	FDFE_FFFF	4G – 48M	4G – 32M – 1	0000_0000–00FF_FFFF	PCI/ISA memory space (16 Mbytes) ³
FE00_0000	FE7F_FFFF	4G – 32M	4G – 24M – 1	0000_0000–007F_FFFF	PCI/ISA I/O space (8 Mbytes), 0-based ⁴
FE80_0000	FEBF_FFFF	4G – 24M	4G – 20M – 1	0080_0000–00BF_FFFF	PCI I/O space (4 Mbytes), 0-based ⁵
FEC0_0000	FEDF_FFFF	4G – 20M	4G – 18M – 1	CONFIG_ADDR	PCI configuration address register ⁶
FEE0_0000	FEFE_FFFF	4G – 18M	4G – 17M – 1	CONFIG_DATA	PCI configuration data register ⁷
FEF0_0000	FEFF_FFFF	4G – 17M	4G – 16M – 1	FEF0_0000–FEFF_FFFF	PCI interrupt acknowledge
FF00_0000	FF7F_FFFF	4G – 16M	4G – 8M – 1	FF00_0000–FF7F_FFFF	32- or 64-bit Flash/ROM space (8 Mbytes) ⁸
FF80_0000	FFFF_FFFF	4G – 8M	4G – 1	FF80_0000–FFFF_FFFF	8-, 32- or 64-bit Flash/ROM space (8 Mbytes) ⁹

Table 4-5. Address Map B—PCI Memory Master View (Host Bridge)

PCI Memory Transaction Address Range				Processor Core Address Range	Definition
Hex		Decimal			
0000_0000	0009_FFFF	0	640K – 1	0000_0000–0009_FFFF	System memory space
000A_0000	000F_FFFF	640K	1M – 1	000A_0000–000F_FFFF	Compatibility hole ¹
0010_0000	3FFF_FFFF	1M	1G – 1	0010_0000–3FFF_FFFF	System memory space
4000_0000	7FFF_FFFF	1G	2G – 1	4000_0000–7FFF_FFFF	Reserved ²
8000_0000	FCFF_FFFF	2G	4G – 48M – 1	No system memory cycle	PCI memory space

Table 4-5. Address Map B—PCI Memory Master View (Host Bridge) (Continued)

PCI Memory Transaction Address Range				Processor Core Address Range	Definition
Hex		Decimal			
FD00_0000	FDFE_FFFF	4G – 48M	4G – 32M – 1	00000000–00FFFFFF	System memory space ¹⁰
FE00_0000	FEFF_FFFF	4G – 32M	4G – 16M – 1	No system memory cycle	Reserved
FF00_0000	FF7F_FFFF	4G – 16M	4G – 8M – 1	FF00_0000–FF7F_FFFF	32- or 64-bit Flash/ROM space (8 Mbytes) ⁸
FF80_0000	FFFF_FFFF	4G – 8M	4G – 1	FF80_0000–FFFF_FFFF	8-, 32- or 64-bit Flash/ROM space (8 Mbytes) ⁹

Table 4-6. Address Map B—PCI Memory Master View (Agent Bridge)

PCI Memory Transaction Address Range	Processor Core Address Range	Definition
(PCSRBAR) - (PCSRBAR + 4 Kbyte)	—	PCI control and status registers
(LMBAR) - (LMBAR + window size)	—	PCI local memory space

Table 4-7. Address Map B—PCI I/O Master View

PCI I/O Transaction Address Range				Processor Core Address Range	Definition
Hex		Decimal			
0000_0000	0000_FFFF	0	64K – 1	No system memory cycle	PCI/ISA I/O space
0001_0000	007F_FFFF	64K	8M – 1	No system memory cycle	Reserved
0080_0000	00BF_FFFF	8M	12M – 1	No system memory cycle	PCI I/O space
00C0_0000	FFFF_FFFF	12M	4G – 1	No system memory cycle	Reserved

Notes

1. This address range is separately programmable (see Section 5.8, “Address Map B Options Register”) for the processor interface and the PCI interface to control whether accesses to this address range go to system memory or PCI memory.
2. The MPC8240 generates a memory select error (if enabled; see Section 5.7.2, “Error Enabling Registers”) for transactions in the address range 4000_0000–7FFF_FFFF. If memory select errors are disabled, the MPC8240 returns all 1s for read operations and no update for write operations.
3. If AMBOR[CPU_FD_ALIAS_EN] = 1 (see Section 5.8, “Address Map B Options Register”), the MPC8240 forwards processor transactions in this range to the zero-based PCI memory space with the 8 most significant bits cleared (that is, AD[31–0] = 0x00 || A[8–31] of the internal peripheral logic address bus).
4. Processor addresses are translated to PCI addresses as follows:
PCI address (AD[31–0]) = 0x00 || A[8–31] to generate an address range 0000_0000–007F_FFFF. Note that the processor address range FE01_0000–FE7F_FFFF is reserved for future use.
5. The MPC8240 forwards processor transactions in this range to the PCI I/O space with the 8 most significant bits cleared (that is, AD[31–0] = 0x00 || A[8–31]).
6. Each word in this address range is aliased to the PCI CONFIG_ADDR register. See Section 5.1.2, “Processor Access to Configuration Registers (Map B).”
7. Each word in this address range is aliased to the PCI CONFIG_DATA register. See Section 5.1.2, “Processor Access to Configuration Registers (Map B).”
8. The processor and PCI masters can access ROM/Flash on the local bus in the address range 0xFF00_0000 - 0xFF7F_FFFF if the ROM/Flash is configured to be on the local bus at reset or if PIRC2[CF_FF0_LOCAL] = 1 (see Section 5.6, “Processor Interface Configuration Registers”); otherwise, the address is sent to PCI. This address range will always be treated as an access to a 32-bit or 64-bit wide device as configured at reset if it is configured to be on the local bus.
9. The processor and PCI masters can access ROM/Flash on the local bus in the address range 0xFF70_0000 - 0xFFFF_FFFF if the ROM/Flash is configured to be on the local bus at reset (see Section 5.6, “Processor Interface Configuration Registers”); otherwise, the address is sent to PCI. This address range will be treated as an access to a 8-bit, 32-bit or 64-bit wide device as configured at reset if it is configured to be on the local bus.
10. If AMBOR[PCI_FD_ALIAS_EN] = 1 (see Section 5.8, “Address Map B Options Register”), the MPC8240 forwards PCI memory transactions in this range to local memory with the 8 most significant bits cleared (that is, 0x00 || AD[23–0]).

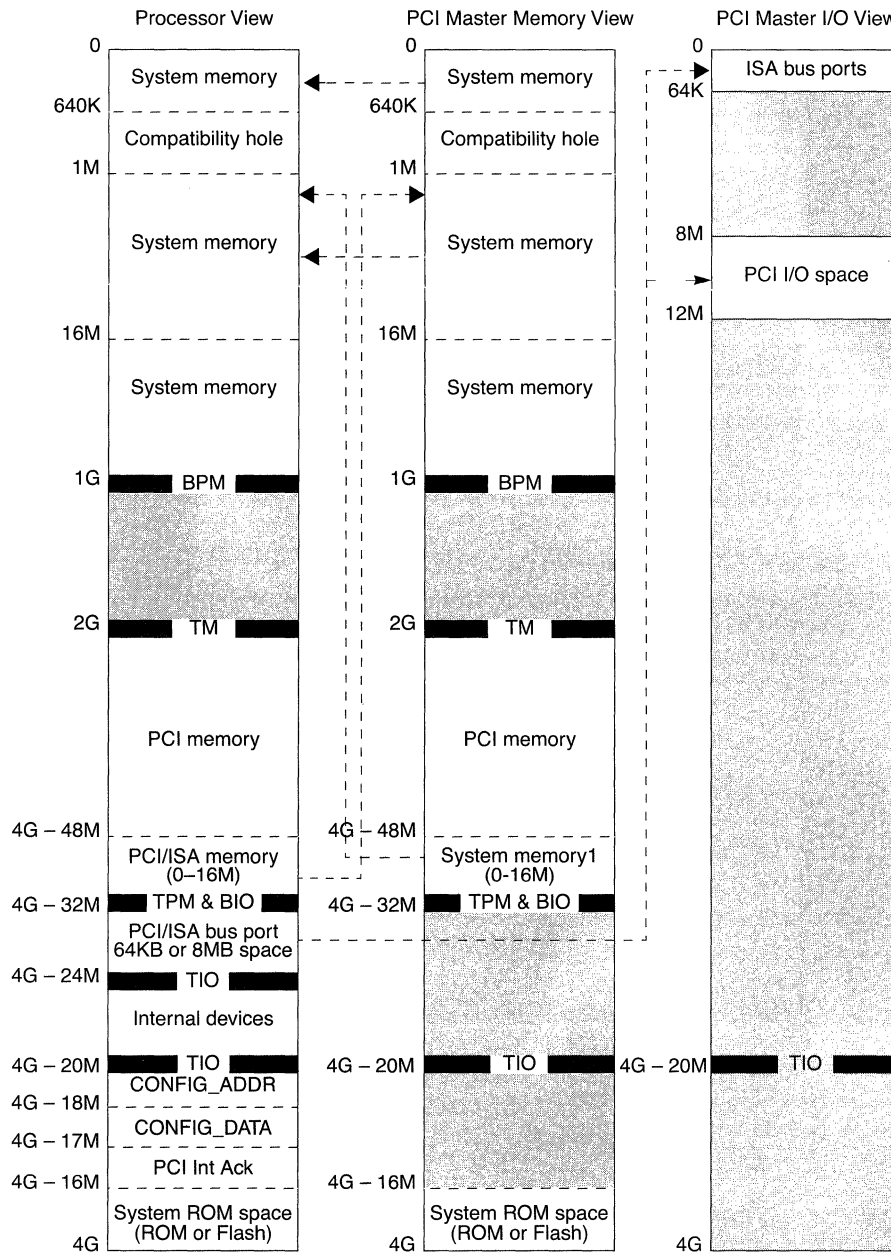


Figure 4-4. Address Map B in Host Mode

4.3 Address Translation

The MPC8240 allows remapping of PCI memory space (inbound) transactions to local memory and processor core (outbound) transactions to PCI memory space. Note that address translation is supported only for agent mode; it is not supported when the MPC8240 is operating in host mode. Also note that since agent mode is supported only for address map B, address translation is supported only for address map B. The following sections describe the address translation support of the MPC8240.

4.3.1 Inbound PCI Address Translation

For inbound address translation, an inbound memory window is specified in the upper 2 Gbytes of PCI memory space and an inbound translation window is specified in the MPC8240 local memory space. PCI memory accesses in the inbound memory window are claimed by the MPC8240 and are forwarded to local memory with the address translated to the inbound translation window. PCI memory transactions outside of the inbound memory window are ignored (not claimed) by the MPC8240.

Figure 4-5 shows inbound PCI address translation from PCI memory space to the local memory space.

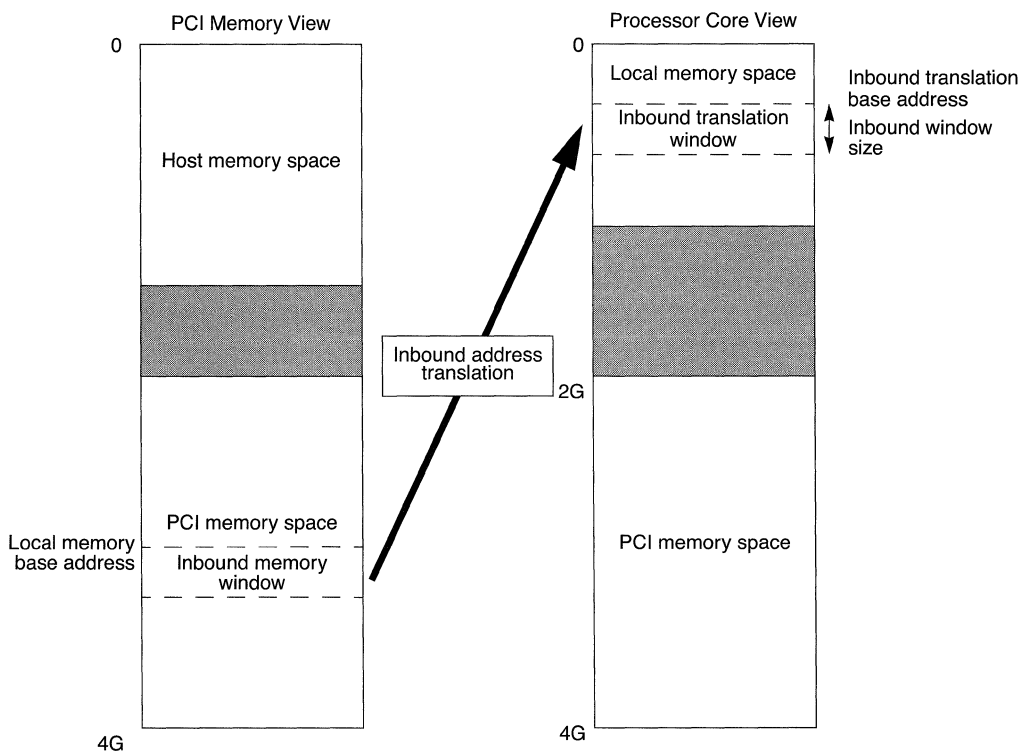


Figure 4-5. Inbound PCI Address Translation

Inbound address translation only allows address translation to the local memory space (the lower 1 Gbyte of the address space). This means that an external PCI master cannot access devices in the ROM/Port X address space when using inbound address translation. Since the local memory space is restricted to addresses below 0x4000_0000 (1 Gbyte), any access that gets translated above 0x4000_0000 triggers a memory select error. Thus, the entire inbound translation window must be programmed to be below 0x4000_0000.

The local memory base address register (LMBAR) and the inbound translation window register (ITWR) specify the location and size of the inbound memory window and the inbound translation window. These registers are described in Section 4.3.3, “Address Translation Registers.” Inbound address translation may be disabled by programming the inbound window size in the ITWR to all zeros.

Note that overlapping the inbound and outbound translation windows is not supported and can cause unpredictable behavior.

4.3.2 Outbound PCI Address Translation

For outbound translation, an outbound memory window is specified in the upper 2 Gbytes of the MPC8240's address space, and an outbound translation window is specified in the PCI memory space. Processor transactions within the outbound memory window are forwarded to the PCI bus with the address translated to the outbound translation window. Outbound transaction addresses outside of the outbound memory window are forwarded to the PCI bus untranslated.

Figure 4-6 shows outbound PCI address translation from the processor core address space to PCI memory space.

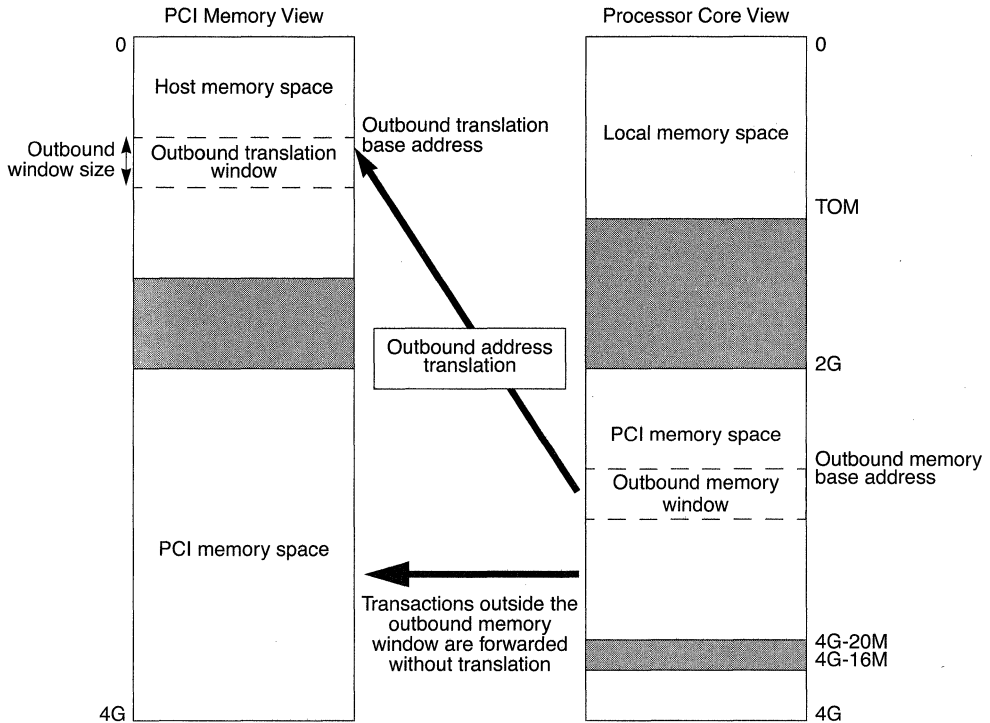


Figure 4-6. Outbound PCI Address Translation

Transactions to the MPC8240 address space marked as configuration address, configuration data, and interrupt acknowledge (0xFEC0_0000—0xFEFF_FFFF) are excluded from the outbound memory window. If the outbound memory base address is set to include this range, the MPC8240 will not translate the accesses to the outbound translation window. That is, the range appears as a hole in the outbound translation.

The outbound memory base address register (OMBAR) and the outbound translation window register (OTWR) specify the location and size of the outbound memory window and the outbound translation window. These registers are described in Section 4.3.3, “Address Translation Registers.” Outbound address translation may be disabled by programming the outbound window size to all zeros.

Note that overlapping the inbound and outbound translation windows is not supported and can cause unpredictable behavior.

4.3.3 Address Translation Registers

This section describes the address translation registers in detail. The address translation registers, summarized in Table 4-8, specify the windows for inbound and outbound address translation.

Table 4-8. ATU Register Summary

Register Name	Location	Description
Local memory base address register (LMBAR)	MPC8240 internal configuration registers—see Chapter 5, “Configuration Registers” Offset 0x10	Specifies the starting address of the inbound memory window. PCI memory transactions in the inbound memory window are translated to the inbound translation window (specified in the ITWR) in local memory.
Inbound translation window register (ITWR)	EUMB—see Section 4.4, “Embedded Utilities Memory Block (EUMB)” Offset 0x0_2310 (local) Offset 0x310 (PCI)	Specifies the starting address of the inbound translation window and the size of the window.
Outbound memory base address register (OMBAR)	EUMB—see Section 4.4, “Embedded Utilities Memory Block (EUMB)” Offset 0x0_2300 (local) Offset 0x300 (PCI)	Specifies the starting address for the outbound memory window. Processor transactions in the outbound memory window are translated to the outbound translation window (specified in the OTWR) in PCI memory space.
Outbound translation window register (OTWR)	EUMB—see Section 4.4, “Embedded Utilities Memory Block (EUMB)” Offset 0x0_2308 (local) Offset 0x308 (PCI)	Specifies the starting address of the outbound translation window and the size of the window.

4.3.3.1 Local Memory Base Address Register (LMBAR)

The LMBAR, shown in Figure 4-7 and Table 4-11, defines the inbound memory window.

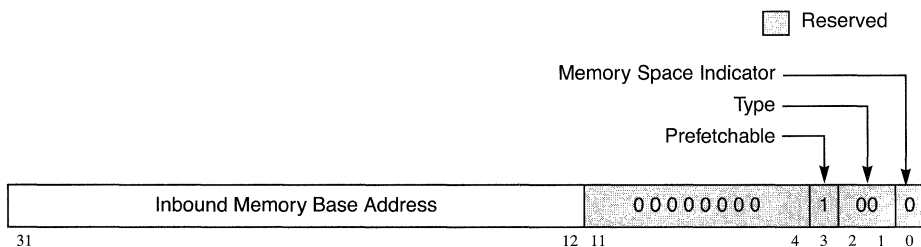


Figure 4-7. Local Memory Base Address Register (LMBAR)—0x10

Table 4-9. Bit Settings for LMBAR—0x10

Bits	Name	Reset Value	R/W	Description
31–12	Inbound memory base address	0x0000_0	R/W	Indicates the base address where the inbound memory window resides. The inbound memory window must be aligned based on the granularity specified by the inbound window sizes specified in the ITWR. Note that the EUMB area must be selected first, then the ITWR programmed, and then these bits can be set. Refer to Chapter 4, “Address Maps,” for more information on the EUMB and the ATU.
11–4	Reserved	All 0s	R	Reserved; the MPC8240 only allows a minimum of a 4KByte window.
3	Prefetchable	1	R	Indicates that the space is prefetchable.

Table 4-10. Bit Settings for ITWR—0x0_2310 (Continued)

Bits	Name	Reset Value	R/W	Description
4–0	Inbound window size	All 0s	R/W	<p>Inbound window size. The inbound window size is encoded as N where the window size is 2^{N+1} bytes. The minimum window size is 4 Kbytes; the maximum window size is 2 Gbytes. Note that the inbound window size sets the size of both the inbound memory window and the inbound translation window.</p> <p>00000 Inbound address translation disabled 00001 Reserved ... 01010 Reserved 01011 2^{12} = 4 Kbyte window size 01100 2^{13} = 8 Kbyte window size 01101 2^{14} = 16 Kbyte window size ... 11101 2^{30} = 1 Gbyte window size 11110 Reserved 11111 Reserved</p>

4.3.3.3 Outbound Memory Base Address Register (OMBAR)

The OMBAR, shown in Figure 4-9 and Table 4-11, defines the outbound memory window.

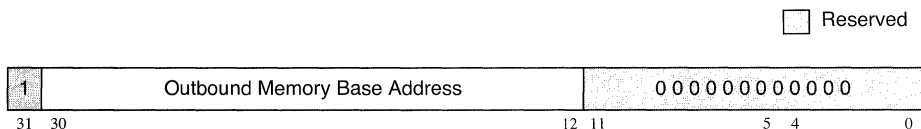


Figure 4-9. Outbound Memory Base Address Register (OMBAR)—0x0_2300

Table 4-11. Bit Settings for OMBAR—0x0_2300

Bits	Name	Reset Value	R/W	Description
31	—	1	R	Reserved. The outbound memory window must reside in the upper 2 Gbytes of the MPC8240 address space.
30–12	Outbound memory base address	Undefined	R/W	Processor address that is the starting address for the outbound memory window. The outbound memory window must be aligned based on the granularity specified by the outbound window sizes specified in the OTWR.
11–0	—	All 0s	R	Reserved

4.3.3.4 Outbound Translation Window Register (OTWR)

The OTWR, shown in Figure 4-10 and Table 4-12, defines the outbound translation window and outbound window size. The outbound window size in the OTWR sets the size of both the outbound translation window in PCI memory space and the outbound memory window in the processor address space. Software can alter the outbound translation base address and the outbound translation window size during runtime. This allows software to scroll through host memory or address alternate space as needed.

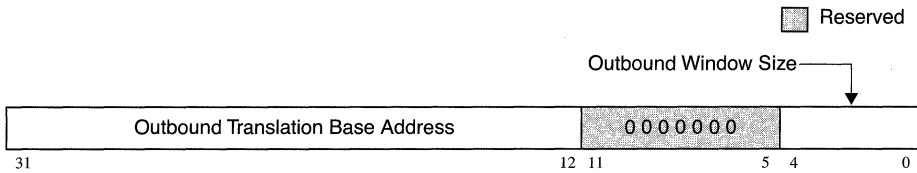


Figure 4-10. Outbound Translation Window Register (OTWR)—0x0_2308

Table 4-12. Bit Settings for OTWR—0x0_2308

Bits	Name	Reset Value	R/W	Description
31–12	Outbound translation base address	Undefined	R/W	PCI memory address—the starting address for the outbound translation window. The outbound translation window must be aligned based on the granularity specified by the outbound window size.
11–5	—	All 0s	R	Reserved
4–0	Outbound window size	All 0s	R/W	Outbound window size—The outbound window size is encoded as N where the window size is 2^{N+1} bytes. The minimum window size is 4 Kbytes; the maximum window size is 2 Gbytes. Note that the outbound window size sets the size of both the outbound memory window and the outbound translation window. 00000 Outbound address translation disabled 00001 Reserved ... 01010 Reserved 01011 2^{12} = 4 Kbyte window size 01100 2^{13} = 8 Kbyte window size 01101 2^{14} = 16 Kbyte window size ... 11101 2^{30} = 1 Gbyte window size 11110 2^{31} = 2 Gbyte window size 11111 Reserved

4.4 Embedded Utilities Memory Block (EUMB)

The MPC8240 contains several embedded features that require control and status registers. These registers are accessible during normal operation. The features include the DMA controller, messaging unit, EPIC, I²C, and ATU. These registers in some cases are accessible by both the processor core and the PCI bus. The collection of these units is called the embedded utilities. A block of local memory and PCI memory space is allocated to these runtime registers. Thus, registers within the EUMB are located from 0x8000_0000 to 0xFDFD_FFFF.

The embedded utilities memory block is relocatable both in PCI memory space (for PCI access) and local memory space (for processor access). The local memory map location of this register block is controlled by the embedded utilities memory block base address register (EUMBBAR); see Section 5.5, “Embedded Utilities Memory Block Base Address

Register.” The PCI bus memory map location for this block is controlled by the peripheral control and status registers base address register (PCSRBAR); see Section 5.2.7, “PCI Base Address Registers.”

Note that the EUMBBAR should not reside inside the outbound translation window. Operation is not guaranteed if the two are overlapping.

All registers in the EUMB are accessible with 32-bit accesses only. Transactions of sizes other than 32-bit are considered a programming error, and operation is not guaranteed.

4.4.1 Processor Core Control and Status Registers

The memory mapped registers of the EUMB that are accessible by the processor for the message unit, DMA controller, ATU, I²C controller and EPIC unit are described in Figure 4-11.

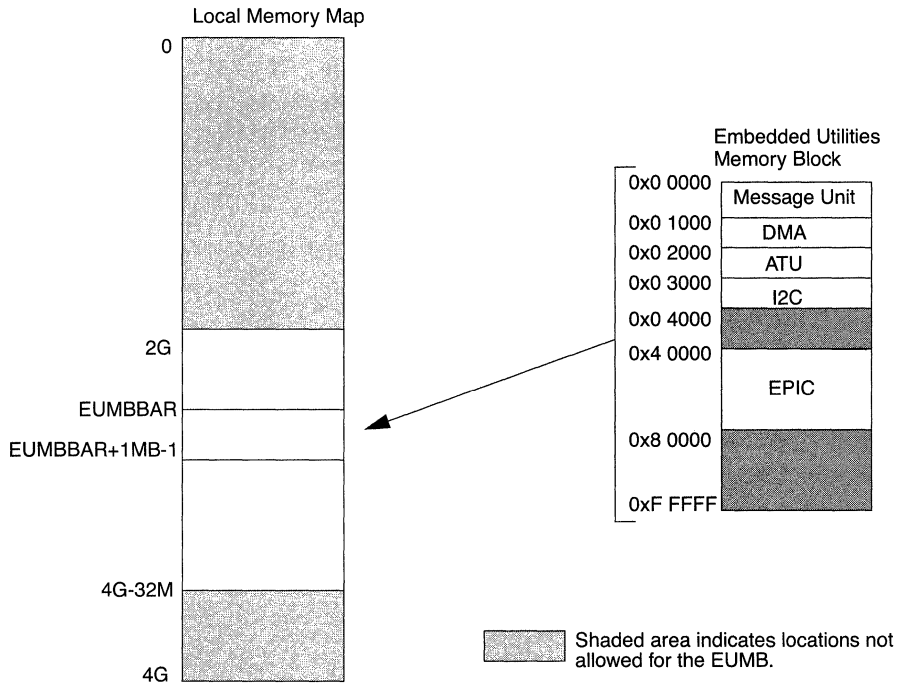


Figure 4-11. Embedded Utilities Memory Block Mapping to Local Memory

Embedded Utilities Memory Block (EUMB)

Table 4-13 summarizes the embedded utilities local memory registers and their offsets.

Table 4-13. Embedded Utilities Local Memory Register Summary

Local Memory Offset	Register Set	Reference
0x0_0000 - 0x0_0FFF	Message registers, Doorbell interface, I ₂ O	Section 10.2, "Message and Doorbell Register Programming Model" Section 10.2.3, "Doorbell Register Descriptions" Section 10.3, "I ₂ O Interface"
0x0_1000 - 0x0_1FFF	DMA controller	Section 9.2, "DMA Register Summary"
0x0_2000 - 0x0_2FFF	ATU	Section 4.3.3, "Address Translation Registers"
0x0_3000 - 0x0_3FFF	I ² C controller	Section 11.3, "Programming Model"
0x0_4000 - 0x3_FFFF	Reserved	—
0x4_0000 - 0x7_FFFF	EPIC controller	Section 12.2, "EPIC Register Summary"
0x8_0000 - 0xF_FFFF	Reserved	—

4.4.2 Peripheral Control and Status Registers

The MPC8240 contains a set of memory mapped registers that are accessible from the PCI bus. These registers allow external masters on the PCI bus to access the MPC8240's onboard embedded utilities such as the message unit and DMA controller as shown in Figure 4-12. The region requires 4 Kbytes of PCI memory space. The base address for this region is selectable through the PCI configuration register PCSRBAR, see Section 5.2.7, "PCI Base Address Registers."

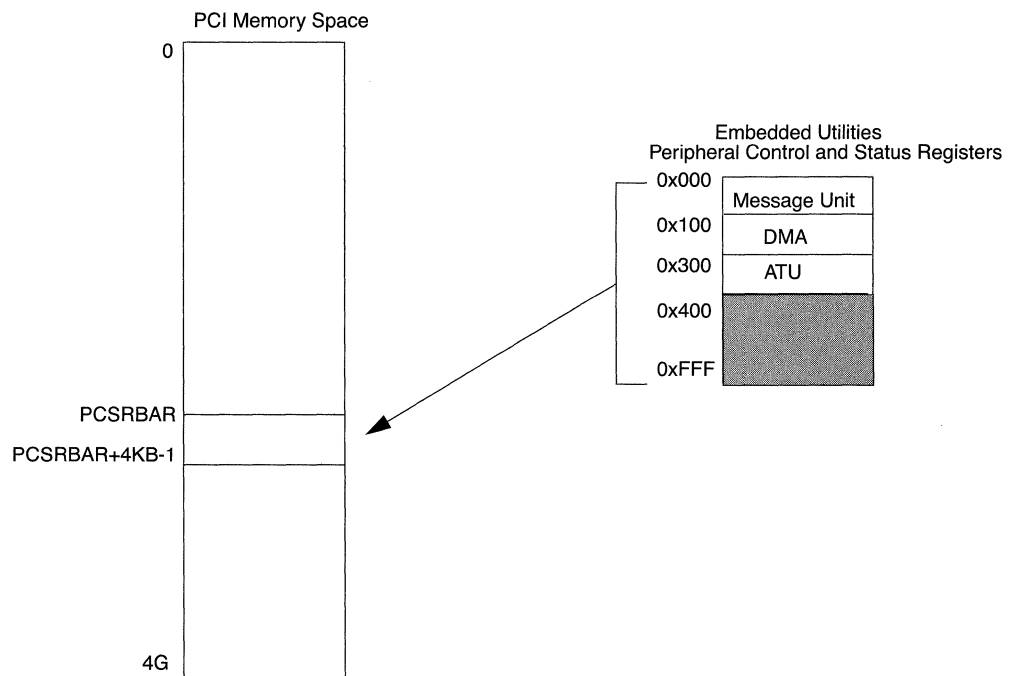


Figure 4-12. Embedded Utilities Memory Block Mapping to PCI Memory

Table 4-14 summarizes the embedded utilities registers accessible by the PCI bus and their offsets.

Table 4-14. Embedded Utilities Peripheral Control and Status Register Summary

PCI Memory Offset	Register Set	Reference
0x000 – 0x0FF	Message registers, Doorbell interface, I ₂ O	Section 10.2, “Message and Doorbell Register Programming Model” Section 10.2.3, “Doorbell Register Descriptions” Section 10.3, “I2O Interface”
0x100 – 0x2FF	DMA	Section 9.2, “DMA Register Summary”
0x300 – 0x3FF	ATU	Section 4.3.3, “Address Translation Registers”
0x400 – 0xFFF	Reserved	—

Embedded Utilities Memory Block (EUMB)

Chapter 5

Configuration Registers

This chapter describes the programmable configuration registers of the MPC8240. These registers are generally set up by initialization software following a power-on reset, hard reset, or error handling routines. All the internal registers of the MPC8240 are intrinsically little-endian. In the register descriptions of this chapter, bit 0 is the least significant bit of the register.

Reserved bits in the register descriptions are not guaranteed to have predictable values. Software must preserve the values of reserved bits when writing to a configuration register. Also, when reading from a configuration register, software should not rely on the value of any reserved bit remaining consistent. Thus, the values of reserved bit positions must first be read, merged with the new values for other bit positions, and then written back. Software should use the transfer size shown in the register bit descriptions throughout this chapter.

5.1 Configuration Register Access

The MPC8240 user-accessible configuration registers are intended for configuration, initialization, and error handling. Their use is intended for initialization software and error handling software.

The MPC8240 configuration registers are accessible from the processor core through a memory-mapped configuration port. In addition, a subset of the configuration registers are accessible from the PCI bus through the use of PCI configuration cycles.

5.1.1 Processor Access to Configuration Registers (Map A)

The processor has access to the Map A configuration registers through direct and indirect access methods as described in the following subsections.

5.1.1.1 Indirect Access Method

When address map A is in use, the MPC8240 configuration registers are accessed by an indirect method similar to accessing PCI device configuration registers. The 32-bit register address `0x8000_00nn`, where `nn` is the address offset of the desired configuration register (see Table 5-1 and Figure 5-1) is written to `CONFIG_ADDR` at `0x8000_0CF8`. Then, the data is accessed at `CONFIG_DAT` at addresses `0x8000_0CFC–0x8000_0CFF`. In map A, the internal configuration registers can only be accessed by this method.

5.1.1.2 Direct Access Method

The MPC8240 allows the generation of a PCI configuration cycle using the 4Kbyte address range starting at 0x8080_xn, where xx are the lines to be used as IDSEL, and nn corresponds to the 256 bytes assigned to the PCI configuration registers. As stated in the PCI specification, only one IDSEL can be asserted for each transaction.

5.1.2 Processor Access to Configuration Registers (Map B)

When using address map B, the MPC8240 uses a similar indirect method to access the internal configuration registers. (The exception is that CONFIG_ADDR and CONFIG_DAT are found at different addresses.) The 32-bit register address—0x8000_00nn, where nn is the address offset of the desired configuration register—is written to CONFIG_ADDR at any word-aligned address in the range (0xFEC0_0000–0xFEDF_FFFF). Every word within this range is aliased to the same location. Then, the data is accessed at CONFIG_DAT at any address in the range 0xFEE0_0000–0xFEEF_FFFF. Every word within this range is aliased to the same location.

5.1.3 PCI Access to Configuration Registers

A subset of the configuration registers are accessible from the PCI bus allowing configuration of the PCI port. The MPC8240 responds to standard PCI configuration cycles when the IDSEL signal is asserted. The listing of PCI accessible configuration registers are found in Table 5-2.

5.1.4 Configuration Register Access in Little-Endian Mode

When the processor and peripheral logic are in little-endian mode, the program should access the configuration registers using the methods described in Section 5.1, “Configuration Register Access.” The data appears in the processor register in descending significance byte order (MSB to LSB) at the time it is stored to the peripheral logic. For the indirect-access method, the configuration register address in the processor register should appear (as data appears) in descending byte order (MSB to LSB) when it is stored to the peripheral logic.

Example: Map A configuration sequence, 4-byte data write to register at address offset 0xA8

```
Initial values:r0 contains 0x8000_00A8
               r1 contains 0x8000_0CF8
               r2 contains 0xAABB_CCDD
Register at 0xA8 contains 0xFFFF_FFFF (AB to A8)
Code sequence: stw    r0,0(r1)
               sync
               stw    r2,4(r1)
               sync
```

```
Results:Address 0x8000_0CF8 contains 0x8000_00A8 (MSB to LSB)
Register at 0xA8 contains 0xAABB_CCDD (AB to A8)
```

Example: Map A configuration sequence, 2-byte data write to register at address offset 0xAA

```
Initial values:r0 contains 0x8000_00A8
               r1 contains 0x8000_0CF8
               r2 contains 0xAABB_CCDD
Register at 0xA8 contains 0xFFFF_FFFF (AB to A8)
```

```
Code sequence: stw    r0,0(r1)
               sync
               sth    r2,6(r1)
               sync
```

Results:Address 0x8000_0CF8 contains 0x8000_00A8 (MSB to LSB)
 Register at 0xA8 contains 0xCCDD_FFFF (AB to A8)

Note that in this example, the value 0x8000_00A8 is the configuration address register, not 0x8000_00AA. The address offset 0xAA is generated by using 0x8000_0CFE for the data access.

Example: Map A configuration sequence, 1-byte data read from register at address offset 0xA9

```
Initial values:r0 contains 0x8000_00A8
               r1 contains 0x8000_0CF8
               Register at 0xA8 contains 0xAABB_CCDD (AB to A8)
```

```
Code sequence: stw    r0,0(r1)
               sync
               lbz    r2,5(r1)
               sync
```

Results:Address 0x8000_0CF8 contains 0x8000_00A8 (MSB to LSB)
 r2 contains 0x0000_00CC

Example: Map B address map configuration sequence, 4-byte data write to register at address offset 0xA8

```
Initial values:r0 contains 0x8000_00A8
               r1 contains 0xFEC0_0000
               r2 contains 0xFEE0_0000
               r3 contains 0xAABB_CCDD
               Register at 0xA8 contains 0xFFFF_FFFF (AB to A8)
```

```
Code sequence: stw    r0,0(r1)
               sync
               stw    r3,0(r2)
               sync
```

Results:Address 0xFEC0_0000 contains 0x8000_00A8 (MSB to LSB)
 Register at 0xA8 contains 0xAABB_CCDD (AB to A8)

Example: Map B address map configuration sequence, 1-byte data write to register at address offset 0xAA

```
Initial values:r0 contains 0x8000_00A8
               r1 contains 0xFEC0_0000
               r2 contains 0xFEE0_0000
               r3 contains 0xAABB_CCDD
               Register at 0xA8 contains 0xFFFF_FFFF (AB to A8)
```

```
Code sequence: stw    r0,0(r1)
               sync
               stb    r3,2(r2)
               sync
```

Results:Address 0xFEC0_0000 contains 0x8000_00A8 (MSB to LSB)
 Register at 0xA8 contains 0xFFDD_FFFF (AB to A8)

5.1.4.1 Configuration Register Access in Big-Endian Mode

In big-endian mode (both the processor core and the peripheral logic), software must byte-swap the data of the configuration register before performing an access. That is, the data appears in the processor register in ascending byte order (LSB to MSB). For indirect access mode (map A or B), software loads the configuration register address and the configuration register data into the processor register in ascending byte order (LSB to MSB).

Note that in the following examples, the data in the configuration register (at 0xA8) is shown in little-endian order. This is because all the internal registers are intrinsically little-endian.

Configuration Register Access

Example: Map A configuration sequence, 4-byte data write to register at address offset 0xA8

```
Initial values:r0 contains 0xA800_0080
               r1 contains 0x8000_0CF8
               r2 contains 0xDDCC_BBAA
               Register at 0xA8 contains 0xFFFF_FFFF (AB to A8)
Code sequence: stw    r0,0(r1)
               sync
               stw    r2,4(r1)
               sync
```

Results:Address 0x8000_0CF8 contains 0x8000_00A8 (MSB to LSB)
Register at 0xA8 contains 0xAABB_CCDD (AB to A8)

Example: Map B address map configuration sequence, 2-byte data write to register at address offset 0xAA

```
Initial values:r0 contains 0xA800_0080
               r1 contains 0xFEC0_0000
               r2 contains 0xFEE0_0000
               r3 contains 0xDDCC_BBAA
               Register at 0xA8 contains 0xFFFF_FFFF (AB to A8)
Code sequence: stw    r0,0(r1)
               sync
               sth    r3,2(r2)
               sync
```

Results:Address 0xFEC0_0000 contains 0x8000_00A8 (MSB to LSB)
Register at 0xA8 contains 0xAABB_FFFF (AB to A8)

5.1.5 Configuration Register Summary

The following sections summarize the addresses and attributes of the configuration registers accessible by both the processor and the PCI interface.

5.1.5.1 Processor-Accessible Configuration Registers

Table 5-1 describes the configuration registers that are accessible by the processor core. Not all of the registers are shown in this document. Note that any configuration addresses not defined in Table 5-1 are reserved.

Table 5-1. MPC8240 Configuration Registers Accessible from the Processor Core

Address	Size	Program Access Size (Bytes)	Register	Register Access	Reset Value
00	2 bytes	2	Vendor ID = 1057H (not shown)	Read	0x1057
02	2 bytes	2	Device ID = 0003H (not shown)	Read	0x0003
04	2 bytes	2	PCI command register	Read/Write	mode dependent 0x0004 host 0x0000 agent
06	2 bytes	2	PCI status register	Read/Bit Reset	0x00A0
08	1 byte	1	Revision ID (not shown)	Read	0xnn
09	1 byte	1	Standard programming interface	Read	mode-dependent 0x00 host 0x01 agent
0A	1 byte	1	Subclass code (not shown)	Read	0x00
0B	1 byte	1	Class code	Read	mode-dependent 0x06 host 0x0E agent
0C	1 byte	1	Cache line size	Read/Write	0x00
0D	1 byte	1	Latency timer	Read/Write	0x00

Table 5-1. MPC8240 Configuration Registers Accessible from the Processor Core

Address	Size	Program Access Size (Bytes)	Register	Register Access	Reset Value
0E	1 byte	1	Header type (not shown)	Read	0x00
0F	1 byte	1	BIST control	Read	0x00
10	4 bytes	4	Local memory base address register	Read/Write	0x0000_0008
14	4 bytes	4	Peripheral control and status register base address register	Read/Write	0x0000_0000
3C	1 byte	1	Interrupt line	Read/Write	0x00
3D	1 byte	1	Interrupt pin (not shown)	Read	0x01
3E	1 byte	1	MIN GNT (not shown)	Read	0x00
3F	1 byte	1	MAX LAT (not shown)	Read	0x00
46	2 bytes	2	PCI arbiter control register	Read/Write	0x0000
70	2 bytes	1 or 2	Power management configuration register 1 (PMCR1)	Read/Write	0x00
72	1 byte	1	Power management configuration register 2 (PMCR2)	Read/Write	0x00
73	1 byte	1	Output driver control register	Read/Write	0xFF
74	2 bytes	1 or 2	CLK driver control register	Read/Write	0x0000
78	4 bytes	4 bytes	Embedded utilities memory block base address register	Read/Write	0x0000_0000
80, 84	4 bytes	1, 2, or 4	Memory starting address registers	Read/Write	0x0000_0000
88, 8C	4 bytes	1, 2, or 4	Extended memory starting address registers	Read/Write	0x0000_0000
90, 94	4 bytes	1, 2, or 4	Memory ending address registers	Read/Write	0x0000_0000
98, 9C	4 bytes	1, 2, or 4	Extended memory ending address registers	Read/Write	0x0000_0000
A0	1 byte	1	Memory bank enable register	Read/Write	0x00
A3	1 byte	1	Page mode counter/timer	Read/Write	0x00
A8	4 bytes	1, 2, or 4	Processor interface configuration 1	Read/Write	0xFF04_0010
AC	4 bytes	1, 2, or 4	Processor interface configuration 2	Read/Write	0x000C_000C
B8	1 byte	1	ECC single bit error counter	Read/Write	0x00
B9	1 byte	1	ECC single bit error trigger register	Read/Write	0x00
C0	1 byte	1	Error enabling register 1	Read/Write	0x01
C1	1 byte	1	Error detection register 1	Read/Bit Reset	0x00
C3	1 byte	1	Processor internal bus error status register	Read/Bit Reset	0x00
C4	1 byte	1	Error enabling register 2	Read/Write	0x00
C5	1 byte	1	Error detection register 2	Read/Bit Reset	0x00
C7	1 byte	1	PCI bus error status register	Read/Bit Reset	0x00
C8	4 byte	1, 2, or 4	Processor/PCI error address register	Read	0x00
E0	1 byte	1,2, or 4	Address map B options register	Read/Write	0x0000_00C3
F0	4 bytes	1, 2, or 4	MCCR1	Read/Write	0xFFn2_0000
F4	4 bytes	1, 2, or 4	MCCR2	Read/Write	0x0000_0000
F8	4 bytes	1, 2, or 4	MCCR3	Read/Write	0x0000_0000
FC	4 bytes	1,2, or 4	MCCR4	Read/Write	0x0000_0000
others	—	—	Reserved	—	—

Note: Reset values marked mode-dependent are defined by whether the MPC8240 is operating in host or agent mode.

Configuration Register Access

				Address Offset (Hex)
Device ID (0x0003)		Vendor ID (0x1057)		00
PCI Status		PCI Command		04
Class Code	Subclass Code	Standard Programming	Revision ID	08
BIST Control	Header Type	Latency Timer	Cache Line Size	0C
Local Memory Base Address Register				10
Peripheral Control and Status Registers Base Address Register				14
MAX LAT	MIN GNT	Interrupt Pin	Interrupt Line	3C
				40
PCI Arbiter Control				44
Output Driver Control	PMCR2	PMCR1		70
Clock Driver Control Register				74
Embedded Utilities Memory Block Base Address Register				78
Memory Starting Address				80
Memory Starting Address				84
Extended Memory Starting Address				88
Extended Memory Starting Address				8C
Memory Ending Address				90
Memory Ending Address				94
Extended Memory Ending Address				98
Extended Memory Ending Address				9C
Memory Page Mode			Memory Bank Enable	A0
				A4
Processor Interface Configuration Register 1				A8
Processor Interface Configuration Register 2				AC
		ECC Single-Bit Trigger	ECC Single-Bit Counter	B8
				BC
60x Bus Error Status		Error Detection 1	Error Enabling 1	C0
PCI Bus Error Status		Error Detection 2	Error Enabling 2	C4
60x/PCI Error Address				C8
			Addr. Map B Options	E0
Memory Control Configuration Register 1				F0
Memory Control Configuration Register 2				F4
Memory Control Configuration Register 3				F8
Memory Control Configuration Register 4				FC

Figure 5-1. Processor Accessible Configuration Space

5.1.5.2 PCI-Accessible Configuration Registers

Table 5-2 lists the subset of configuration registers that are accessible from the PCI bus. Note that configuration addresses not defined in Table 5-2 are reserved.

Table 5-2. MPC8240 Configuration Registers Accessible from the PCI Bus

Offset	Size	Program access size (Bytes)	Register	Register Access	Reset Value
00	2-bytes	2	Vendor ID = 1057H	Read	0x1057
02	2-bytes	2	Device ID = 0003H	Read	0x0003
04	2-bytes	2	PCI command register	Read/Write	mode-dependent 0x0004 host 0x0000 agent
06	2-bytes	2	PCI status register	Read/Bit-Reset	0x00A0
08	1-byte	1	Revision ID	Read	0xnn
09	1-byte	1	Standard programming interface	Read	mode-dependent 0x00 host 0x01 agent
0A	1-byte	1	Subclass code	Read	0x00
0B	1-byte	1	Class code	Read	mode-dependent 0x06 host 0x0E agent
0C	1-byte	1	Cache line size	Read/Write	0x00
0D	1-byte	1	Latency timer	Read/Write	0x00
0E	1-byte	1	Header type	Read	0x00
0F	1-byte	1	BIST control	Read	0x00
10	4-bytes	4	Local memory base address register	Read/Write	0x0000_0008
14	4-bytes	4	Peripheral control and status register base address register	Read/Write	0x0000_0000
3C	1-byte	1	Interrupt line	Read/Write	0x00
3D	1-byte	1	Interrupt pin	Read	0x01
3E	1-byte	1	MIN GNT	Read	0x00
3F	1-byte	1	MAX LAT	Read	0x00
46	2-bytes	2	PCI arbiter control register	Read/Write	0x0000
others	—	—	Reserved	—	—

Note: Reset values marked mode-dependent are defined by whether the MPC8240 is operating in host or agent mode.

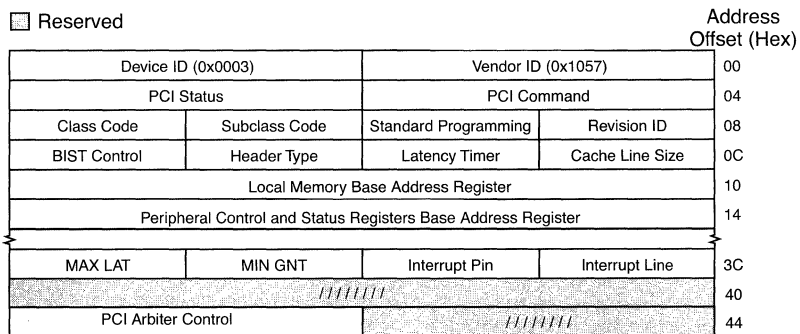


Figure 5-2. PCI Accessible Configuration Space

5.1.6 Configuration Register Order

The configuration registers are described in detail in this chapter in the following order:

- PCI interface configuration registers
- Peripheral logic power management registers
- Output driver configuration registers
- Embedded utilities memory block base address register
- Processor interface configuration registers
- Error handling registers
- Address map B options register
- Memory interface configuration registers

Note that this is roughly the order that these registers appear in the memory map. For example, there are memory configuration registers that appear in the memory map before the processor configuration registers, but they are described in the section on memory interface configuration registers later in the chapter.

5.2 PCI Interface Configuration Registers

The *PCI Local Bus Specification* defines the configuration registers from 0x00 through 0x3F. Table 5-3 summarizes the PCI configuration registers of the MPC8240. Detailed descriptions of these registers are provided in the *PCI Local Bus Specification*.

Table 5-3. PCI Configuration Space Header Summary

Address Offset	Register Name	Description
0x00	Vendor ID	Identifies the manufacturer of the device (0x1057 = Motorola)
0x02	Device ID	Identifies the particular device (0x0003 = MPC8240)
0x04	PCI command	Provides coarse control over a device's ability to generate and respond to PCI bus cycles (see Section 5.2.1, "PCI Command Register," for more information)
0x06	PCI status	Records status information for PCI bus-related events (see Section 5.2.2, "PCI Status Register," for more information)
0x08	Revision ID	Specifies a device-specific revision code (assigned by Motorola)
0x09	Standard programming interface	Identifies the register-level programming interface of the MPC8240 (0x00)
0x0A	Subclass code	Identifies more specifically the function of the MPC8240 (0x00 = host bridge)
0x0B	Base class code	Broadly classifies the type of function the MPC8240 performs (0x06 = bridge device)
0x0C	Cache line size	Specifies the system cache line size

Table 5-3. PCI Configuration Space Header Summary (Continued)

Address Offset	Register Name	Description
0x0D	Latency timer	Specifies the value of the latency timer for this bus master in PCI bus clock units
0x0E	Header type	Bits 0–6 identify the layout of bytes 10–3F; bit 7 indicates a multifunction device. The MPC8240 uses the most common header type (0x00).
0x0F	BIST control	Optional register for control and status of built-in self test (BIST)
0x10–0x33	—	Reserved on the MPC8240
0x34–0x3B	—	Reserved for future use by PCI
0x3C	Interrupt line	Contains interrupt line routing information
0x3D	Interrupt pin	Indicates which interrupt pin the device (or function) uses (0x00 = no interrupt pin)
0x3E	MIN GNT	Specifies the length of the device's burst period (0x00 indicates that the MPC8240 has no major requirements for the settings of latency timers.)
0x3F	MAX LAT	Specifies how often the device needs to gain access to the PCI bus (0x00 indicates that the MPC8240 has no major requirements for the settings of latency timers)
0x43	—	Reserved on the MPC8240

System software may need to scan the PCI bus to determine what devices are actually present. To do this, the configuration software must read the vendor id in each possible PCI slot. If there is no response to a read of an empty slot, the MPC8240 returns 0xFFFF (the invalid vendor id). Any configuration write cycle to a reserved register is completed normally and the data is discarded.

5.2.1 PCI Command Register

The following subsections describe the MPC8240 PCI configuration registers in detail.

The 2-byte PCI command register, shown in Figure 5-3, provides control over the ability to generate and respond to PCI cycles. Table 5-4 describes the bits of the PCI command register.

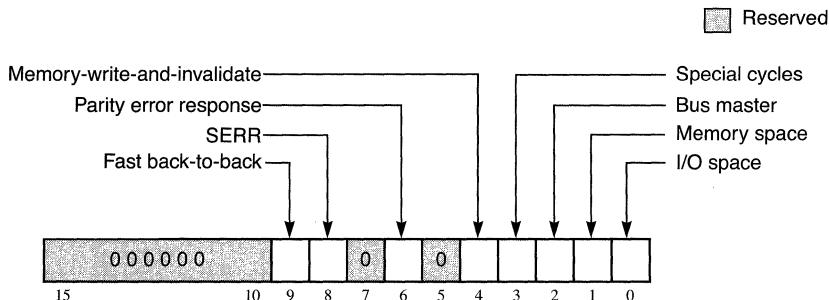


Figure 5-3. PCI Command Register

Table 5-4. Bit Settings for PCI Command Register—0x04

Bits	Name	Reset Value	Description
15–10	—	All 0s	These bits are reserved.
9	Fast back-to-back	0	This bit is hardwired to 0, indicating that the MPC8240 does not run fast back-to-back transactions.
8	SERR	0	This bit controls the SERR driver of the MPC8240. This bit (and bit 6) must be set to report address parity errors. 0 Disables the SERR driver 1 Enables the SERR driver
7	—	0	This bit is reserved.
6	Parity error response	0	This bit controls whether the MPC8240 responds to parity errors. 0 Parity errors are ignored and normal operation continues. 1 Action is taken on a parity error. See Chapter 13, “Error Handling and Exceptions,” for more information.
5	—	0	This bit is reserved.
4	Memory-write-and-invalidate	0	This bit enables generation of the memory-write-and-invalidate command by the MPC8240 as a master. 0 Memory-write command used by MPC8240. 1 Memory-write-and-invalidate command used by MPC8240.
3	Special cycles	0	This bit is hardwired to 0, indicating that the MPC8240 (as a target) ignores all special-cycle commands.

Table 5-4. Bit Settings for PCI Command Register—0x04 (Continued)

Bits	Name	Reset Value	Description
2	Bus master	1 (host) 0 (agent)	This bit controls whether the MPC8240 can act as a master on the PCI bus. Note that if this bit is cleared, processor-to-PCI writes cause the data to be held until it is enabled. Processor to PCI reads with master disabled cause a machine check exception (if enabled). 0 Disables the ability to generate PCI accesses 1 Enables the MPC8240 to behave as a PCI bus master
1	Memory space	0	This bit controls whether the MPC8240 (as a target) responds to memory accesses. 0 The MPC8240 does not respond to PCI memory space accesses. 1 The MPC8240 responds to PCI memory space accesses.
0	I/O space	0	This bit is hardwired to 0, indicating that the MPC8240 (as a target) does not respond to PCI I/O space accesses.

5.2.2 PCI Status Register

The 2-byte PCI status register, shown in Figure 5-4, is used to record status information for PCI bus-related events. The definition of each bit is given in Table 5-5. Only 2-byte accesses to address offset 0x06 are allowed.

Reads to this register behave normally. Writes are slightly different in that bits can be cleared, but not set. A bit is cleared whenever the register is written, and the data in the corresponding bit location is a 1. For example, to clear bit 14 and not affect any other bits in the register, write the value 0b0100_0000_0000_0000 to the register.

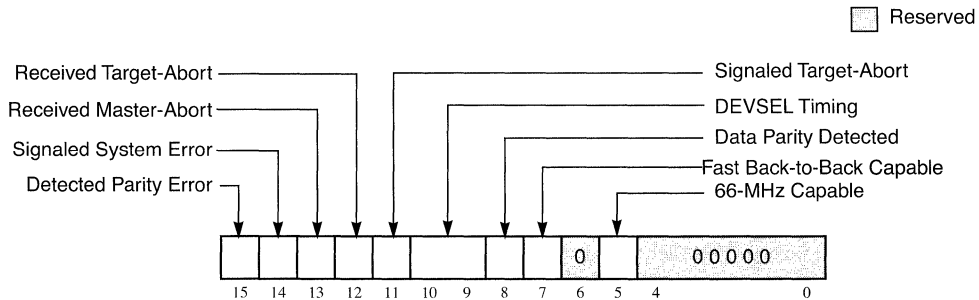


Figure 5-4. PCI Status Register

Table 5-5 describes the bit settings for the PCI status register.

Table 5-5. Bit Settings for PCI Status Register—0x06

Bit	Name	Reset Value	Description
15	Detected parity error	0	This bit is set whenever the MPC8240 detects an address or data parity error, even if parity error handling is disabled (as controlled by bit 6 in the PCI command register).
14	Signaled system error	0	This bit is set whenever the MPC8240 asserts \overline{SERR} .
13	Received master-abort	0	This bit is set whenever the MPC8240, acting as the PCI master, terminates a transaction (except for a special-cycle) using master-abort.
12	Received target-abort	0	This bit is set whenever an MPC8240-initiated transaction is terminated by a target-abort.
11	Signaled target-abort	0	This bit is set whenever the MPC8240, acting as the PCI target, issues a target-abort to a PCI master.
10–9	DEVSEL timing	00	These bits are hardwired to 0b00, indicating that the MPC8240 uses fast device select timing.
8	Data parity detected	0	This bit is set upon detecting a data parity error. Three conditions must be met for this bit to be set: <ul style="list-style-type: none"> • The MPC8240 detected a parity error. • MPC8240 was acting as the bus master for the operation in which the error occurred. • Bit 6 (parity error response) in the PCI command register was set.
7	Fast back-to-back capable	1	This bit is hardwired to 1, indicating that the MPC8240 (as a target) is capable of accepting fast back-to-back transactions.
6	—	0	This bit is reserved.
5	66-MHz capable	1	This bit is read-only and indicates that the MPC8240 is capable of 66-MHz PCI bus operation.
4–0	—	0_0000	These bits are reserved.

5.2.3 Programming Interface

Table 5-6 describes the PCI Programming Interface Register (PIR).

Table 5-6. Programming Interface—0x09

Bit	Reset Value	Description
msb 7–0	Mode-dependent	0x00 When MPC8240 is configured as host bridge 0x01 When MPC8240 is configured as an agent device to indicate the programming model supports the I ₂ O interface

5.2.4 PCI Base Class Code

Table 5-7 describes the PCI Base Class Code Register (PBCCR).

Table 5-7. PCI Base Class Code—0x0B

Bit	Reset Value	Description
msb 7–0	Mode-dependent	0x06 When MPC8240 is configured as a host bridge to indicate “Host Bridge.” 0x0E When MPC8240 is configured as a target device to indicate the device is an agent and is I ₂ O capable.

5.2.5 PCI Cache Line Size

Table 5-8 describes the Processor Cache Line Size Register (PCLSR.)

Table 5-8. Cache Line Size Register—0x0C

Bit	Reset Value	Description
msb 7–0	0x00	Represents the cache line size of the processor in terms of 32-bit words (eight 32-bit words = 32 bytes). This register is read-write; however, an attempt to program this register to any value other than 8 results in setting it to 0.

5.2.6 Latency Timer

Table 5-9 describes the PCI Latency Timer Register (PLTR).

Table 5-9. Latency Timer Register—0x0D

Bit	Reset Value	Description
msb 7–3	00000	The maximum number of PCI clocks for which the device, which is mastering a transaction, will hold the bus after the PCI bus grant has been negated—The value is in PCI clocks. Refer to the PCI 2.1 specification for the rules by which the PCI bus interface unit completes transactions when the timer has expired.
2–0	000	Read-only bits—The minimum latency timer value when set is 8 PCI clocks.

5.2.7 PCI Base Address Registers

Two base address registers are provided when the MPC8240 is used in the PCI agent mode:

- Local Memory Base Address Register (LMBAR)
- Peripheral Control and Status Registers Base Address Register (PCSRBAR)

These registers allow a host processor to configure the base addresses of the MPC8240 when the MPC8240 is being used as a PCI agent. The use of these memory spaces is optional and therefore selectable by the processor. These registers default to read-only, indicating that no memory space is to be configured in the system PCI memory map. It is expected that the local processor core configures the local memory and enables the embedded utilities prior to the host software being allowed to complete PCI configuration.

Table 5-10 describes the bits of the LMBAR.

Table 5-10. Local Memory Base Address Register Bit Definitions—0x10

Bits	Name	Reset Value	R/W	Description
31–12	Inbound memory base address	0x0000_0	R/W	Indicates the base address where the inbound memory window resides. The inbound memory window must be aligned based on the granularity specified by the inbound window sizespecified in the ITWR. Note that the EUMB area must be selected first, then the ITWR programmed, and then the bits set. Refer to Chapter 4, “Address Maps,” for more information on the EUMB and the ATU.
11–4	Reserved	All 0s	R	Reserved; the MPC8240 only allows a minimum of a 4-KByte window.
3	Prefetchable	1	R	Indicates that the space is prefetchable.
2–1	Type	00	R	The inbound memory window may be located anywhere within the 32-bit PCI address space.
0	Memory space indicator	0	R	Indicates PCI memory space

Table 5-11 describes the PCSRBAR.

Table 5-11. PCSR Base Address Register Bit Definitions—0x14

Bit	Reset Value	Read/Write	Description
msb 31–12	0x0000_0	R/W	Indicates the PCI base address that is mapped to the runtime registers (for example, DMA, I ₂ O).
11–0	0x000	R	Reserved

5.2.8 PCI Interrupt Line

Table 5-12 describes the PCI Interrupt Line Register (ILR).

Table 5-12. Interrupt Line Register—0x3C

Bit	Reset Value	Description
msb 7–0	0x00	Contains the interrupt routing information. Software can use this register to hold information regarding on which input of the system interrupt controller corresponds to the INTA signal. Values in this register are system-architecture-specific.

5.2.9 PCI Arbiter Control Register

This register controls the built-in PCI arbitration logic. As many as five external devices are supported.

Table 5-13. PCI Arbiter Control Register Bit Definitions—0x46

Bit	Reset Value	Read/Write	Description
msb 15	x	R/W	Enable internal PCI arbitration 0 If cleared, the internal arbiter is disabled, and the MPC8240 presents its request on $\overline{GNT0}$ to the external arbiter and receives its grant on $\overline{REQ0}$. 1 If set, indicates the internal arbiter is enabled.
14–13	00	R/W	Parking mode controls which device receives the bus grant when there are no outstanding bus requests and the bus is idle. 00 The bus is parked with the last device to use the bus. 01 The bus is parked with the device using $\overline{REQ0}$ and $\overline{GNT0}$. 10 The bus is parked with MPC8240. 11 Reserved; do not use.
12–11	00	R	Reserved
10	0	R/W	Retry PCI Configuration Cycle 1 PCI target logic retries all external PCI configuration transactions. 0 PCI target logic responds to external PCI configuration transactions.
9–8	00	R	Reserved
7	0	R/W	MPC8240 priority level, 1 = high, 0 = low
6–5	00	R	Reserved
4–0	0_0000	R/W	External device priority levels, 1 = high, 0 = low. Bit 0 corresponds to the device using $\overline{REQ0}$ and $\overline{GNT0}$, bit 1 to $\overline{REQ1}$ and $\overline{GNT1}$, etc.

5.3 Peripheral Logic Power Management Configuration Registers (PMCRs)

The power management configuration registers (PMCRs) control the power management functions of the peripheral logic. For more information on the power management feature of both the processor core and the peripheral logic, see Chapter 14, “Power Management.”

5.3.1 Power Management Configuration Register 1 (PMCR1)

Power management configuration register 1 (PMCR1), shown in Figure 5-5, is a 2-byte register located at offset 0x70.

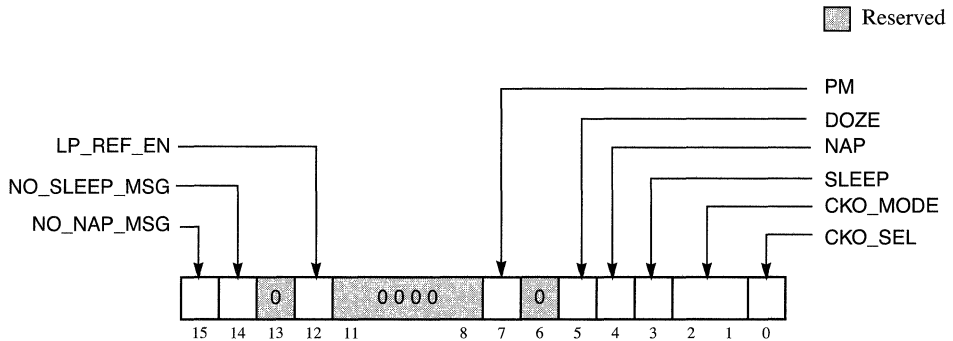


Figure 5-5. Power Management Configuration Register 1 (PMCR1)

Table 5-14 describes the bits of PMCR1.

Table 5-14. Bit Settings for Power Management Configuration Register 1 — 0x70

Bit	Name	Reset Value	Description
15	NO_NAP_MSG	0	HALT command broadcast—Not supported on the MPC8240. 1 Initialization software must set this bit, indicating that the MPC8240 does not broadcast a HALT command on the PCI bus before entering the nap mode.
14	NO_SLEEP_MSG	0	Sleep message broadcast.—Not supported on the MPC8240. 1 Initialization software must set this bit, indicating that the MPC8240 does not broadcast a sleep message command on the PCI bus before entering the sleep mode.
13	—	0	Reserved
12	LP_REF_EN	0	Low-power refresh 0 Indicates that the MPC8240 does not perform memory refresh cycles when it is in sleep mode 1 Indicates that the MPC8240 continues to perform memory refresh cycles when in sleep mode
11–8	—	0	Reserved
7	PM	0	Power management enable 0 Disables the peripheral logic power management logic within the MPC8240 1 Enables the peripheral logic power management logic within the MPC8240
6	—	0	Reserved
5	DOZE	0	Enables/disables the doze mode capability of the MPC8240. Note that this bit is only valid if MPC8240 power management is enabled. (PMCR1[PM] = 1). 0 Disables the doze mode 1 Enables the doze mode

Table 5-14. Bit Settings for Power Management Configuration Register 1—0x70 (Continued)

Bit	Name	Reset Value	Description
4	NAP	0	Enables/disables the nap mode capability of the MPC8240. Note that this bit is only valid if MPC8240 power management is enabled. (PMCR1[PM] = 1). 0 Disables the nap mode 1 Enables the nap mode
3	SLEEP	0	Enables/disables the sleep mode capability of the MPC8240. Note that this bit is only valid if MPC8240 power management is enabled (PMCR1[PM] = 1). 0 Disables the sleep mode 1 Enables the sleep mode
2–1	CKO_MODE	00	Selects the clock source for the test clock output when CKO_SEL = 1. 00 Disables the test clock output driver 01 Selects the internal sys_logic-clk signal as the test clock output source 10 Selects one-half of the PCI rate clock as the test clock output source 11 Selects the internal PCI rate clock as the test clock output source
0	CKO_SEL	x	The initial value of this bit is determined by the \overline{AS} reset configuration bit, which selects either the clock output of the processor core or the clock output of the system logic to be driven out of the CKO signal. 0 Processor core clock selected. The signal driven by CKO is determined by HID0[ECLK,SBCLK]. See Section 2.3.1.2.1, “Hardware Implementation-Dependent Register 0 (HID0),” for the options available. Note that one of the options is for the CKO output driver to be disabled. 1 Output the system logic clock; see the encoding of the CKO_MODE bits above for the available choices.

5.3.2 Power Management Configuration Register 2 (PMCR2)

Power management configuration register 2 (PMCR2), shown in Figure 5-6, is a 1-byte register located at offset 0x72.

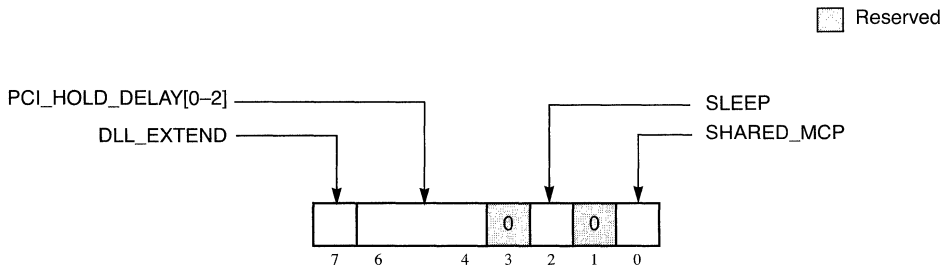


Figure 5-6. Power Management Configuration Register 2 (PMCR2)

Table 5-15 describes the bits of PMCR2.

Table 5-15. Power Management Configuration Register 2—0x72

Bit	Name	Reset Value	Description
msb 7	DLL_EXTEND	0	This bit can be used to shift the lock-range of the DLL by half of a PCI clock cycle.
6–4	PCI_HOLD_DEL	xx0	<p>PCI output hold delay value relative to the PCI_SYNC_IN signal. See the MPC8240 Hardware Specification for the detailed number of nanoseconds guaranteed for each setting. There are eight sequential settings for this value; each corresponds to a set increase in hold time:</p> <p>000 Recommended for 66MHz PCI bus 001 010 011 100 Recommended for 33MHz PCI bus 101 110 Default if reset configuration pins left unconnected 111</p> <p>The initial values of bits 6 and 5 are determined by the reset configuration pins \overline{MCP} and \overline{CKE}, respectively. As these two pins have internal pull-up resistors, the default value after reset is 0b110.</p>
3	—	0	Reserved
2	SLEEP	0	<p>PLL sampling when waking from sleep mode</p> <p>0 The MPC8240 does not sample the PLL configuration pins 1 The MPC8240 samples the PLL configuration pins</p>
1	—	0	Reserved
0	SHARED_MCP	0	<p>0 The \overline{MCP} output is always driven (asserted if there is an error to report; negated otherwise) by the MPC8420. 1 The \overline{MCP} output signal is tri-stated when there is no error to report by the MPC8240.</p>

5.4 Output Driver Configuration Registers

Table 5-16 describes the general output driver control available with the MPC8240 through the Output Driver Control Register (ODCR), and Table 5-17 describes the output driver capability available for the clock signals. Output driver control allows for impedance matching of electrical signals. When driving a capacitive load and the polarity of the driving signal is reversed, the maximum current driven by the output driver of a pin occurs during the transition of the signal. Configure the output driver strength to match the load impedance. The matched impedance limits the maximum current driven during signal transitions. The transition current and mismatched impedance cause ringing on the signal. If the driver level is set too strong, the ringing intensifies. For more information on the output driver type for each signal, refer to the MPC8240 Hardware Specification.

Table 5-16. Output Driver Control Register Bit Definitions—0x73

Bit	Name	Reset Value	Description
msb 7 addr<73>	DRV_PCI	x	Driver capability for PCI and EPIC controller output signals. The initial value of this bit is determined by the reset configuration pin PMAA2. 0 High drive capability on PCI signals (25 Ω) 1 Medium drive capability on PCI signals (50 Ω)
6	DRV_STD	1	Driver capability for standard signals (SDA, SCL, CKO, MCP, Test 4, MIV, and PMAA[0–2]). 0 High drive capability on standard signals (20 Ω) 1 Medium drive capability on standard signals (40 Ω)
5	DRV_MEM_CTRL_1	x	Driver capability for data bus (including QACK and MAA[0–2] signals). Controlled in combination with DRV_MEM_CTRL_2, as follows: 1 20-Ω data bus drive capability; when this is selected, only 8-Ω or 13.3-Ω drive capability allowed for DRV_MEM_CTRL_2 0 40-Ω data bus drive capability, when this is selected, only 20-Ω or 40-Ω drive capability allowed for DRV_MEM_CTRL_2
4	DRV_MEM_CTRL_2	x	Driver capability for address signals (PAR[0–7], FAS[0–7], CAS[0–7], WE, FOE, RCS0, RCS1, SDMA0, SDBA0, AR[1–12], SDRAS, SDCAS, CKE, AS, and SDMA[1–12]). The meaning of this bit setting depends on the setting of DRV_MEM_CTRL_1. The two bits and the meaning of their combined settings for the address signals are shown below: DRV_MEM_CTRL_[1–2] 11 8-Ω drive capability 10 13.3-Ω drive capability 01 20-Ω drive capability 00 40-Ω drive capability The initial value of DRV_MEM_CTRL[1–2] is determined by the PMAA0 and PMAA1 reset configuration pins, respectively.
3	DRV_PCI_CLK_1	1	Driver capability is controlled in combination with DRV_PCI_CLK_2 as shown.
2	DRV_PCI_CLK_2	1	Driver capability is controlled in combination with DRV_PCI_CLK_1, and controls drive strength of PCI_CLK[0–4] and PCI_CLK_SYNC_OUT. DRV_PCI_CLK_[1–2] 11 8-Ω drive capability 10 13.3-Ω drive capability 01 20-Ω drive capability 00 40-Ω drive capability

Table 5-16. Output Driver Control Register Bit Definitions—0x73 (Continued)

Bit	Name	Reset Value	Description
1	DRV_MEM_CLK_1	1	Driver capability is controlled in combination with DRV_MEM_CLK_2 as shown.
0	DRV_MEM_CLK_2	1	Driver capability is controlled in combination with DRV_MEM_CLK_1, and controls drive strength of SDRAM_CLK[0–3] and SDRAM_SYNC_OUT. DRV_MEM_CLK_[1–2] 11 8-Ω drive capability 10 13.3-Ω drive capability 01 20-Ω drive capability 00 40-Ω drive capability

Table 5-17. CLK Driver Control Register Bit Definitions—0x74

Bit	Name	Reset Value	Description
15 addr<75>	—	x	Reserved
14	PCI_CLK0_DIS	0	This bit disables/enables the PCI_CLK0 output of MPC8420. A value of one (0b1) disables the output. A value of zero (0b0) enables the output.
13	PCI_CLK1_DIS	0	This bit disables/enables the PCI_CLK1 output of MPC8420. A value of one (0b1) disables the output. A value of zero (0b0) enables the output.
12	PCI_CLK2_DIS	0	This bit disables/enables the PCI_CLK2 output of MPC8420. A value of one (0b1) disables the output. A value of zero (0b0) enables the output.
11	PCI_CLK3_DIS	0	This bit disables/enables the PCI_CLK3 output of MPC8420. A value of one (0b1) disables the output. A value of zero (0b0) enables the output.

Table 5-17. CLK Driver Control Register Bit Definitions—0x74 (Continued)

Bit	Name	Reset Value	Description
10	PCI_CLK4_DIS	0	This bit disables/enables the PCI_CLK4 output of MPC8420. A value of one (0b1) disables the output. A value of zero (0b0) enables the output.
9–8	—	00	Reserved
7 addr<74>	—	x	Reserved
6	SDRAM_CLK0_DIS	0	This bit disables/enables the SDRAM_CLK0 output of MPC8420. A value of one (0b1) disables the output. A value of zero (0b0) enables the output.
5	SDRAM_CLK1_DIS	0	This bit disables/enables the SDRAM_CLK1 output of MPC8420. A value of one (0b1) disables the output. A value of zero (0b0) enables the output.
4	SDRAM_CLK2_DIS	0	This bit disables/enables the SDRAM_CLK2 output of MPC8420. A value of one (0b1) disables the output. A value of zero (0b0) enables the output.
3	SDRAM_CLK3_DIS	0	This bit disables/enables the SDRAM_CLK3 output of MPC8420. A value of one (0b1) disables the output. A value of zero (0b0) enables the output.
2–0	—	000	Reserved.

5.5 Embedded Utilities Memory Block Base Address Register

The embedded utilities memory block base address register (EUMBBAR), shown in Table 5-18, controls the placement of the embedded utilities memory block (EUMB). See Section 4.4, “Embedded Utilities Memory Block (EUMB).”

Table 5-18. Embedded Utilities Memory Base Address Register—0x78

Bit	Name	Reset Value	Description
msb 31– 20	Base Address	0x000	Base address of the embedded memory utilities block. The block size is 1 Mbyte, and its base address is aligned naturally to a 1 Mbyte address boundary (so the base address is 0xFFFF0_0000). This block is used by processor-initiated transactions and should be located within PCI memory space. Valid values are 0x800–0xFDF. Otherwise, the EUMB is effectively disabled. Thus, registers within the EUMB are located from 0x8000_0000 to 0xFDFE_FFFF.
19–0	—	0x0_0000	Reserved

5.6 Processor Interface Configuration Registers

The processor interface configuration registers (PICRs) control the programmable parameters of the peripheral bus interface to the processor core. There are two 32-bit PICRs—PICR1 and PICR2. Figure 5-7 shows the bits of PICR1.

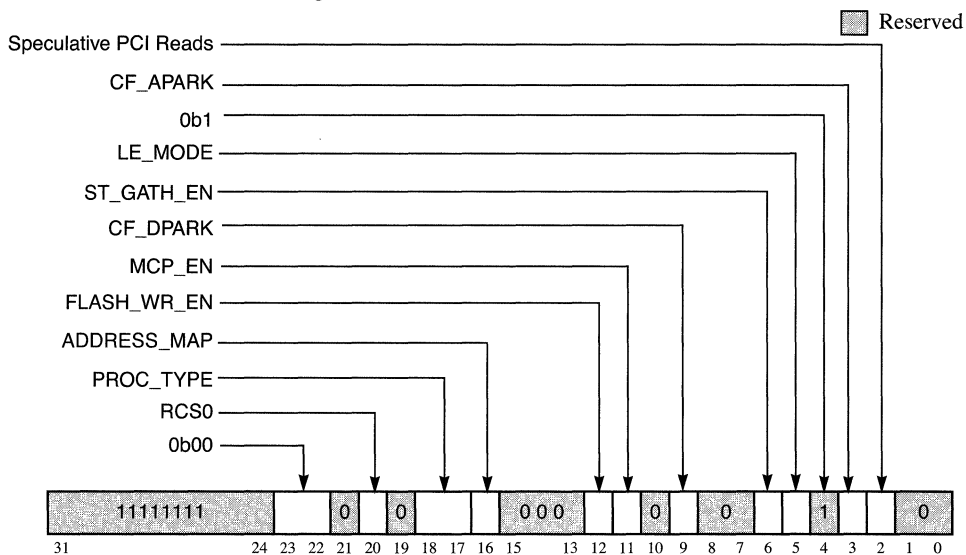


Figure 5-7. Processor Interface Configuration Register 1 (PICR1)—0xA8

Table 5-19 describes the PICR1 bit settings.

Table 5-19. Bit Settings for PICR1—0xA8

Bit	Name	Reset Value	Description
31–24	—	All 1s	Reserved
23–22	—	00	00 Must be cleared to 0b00
21	—	0	Reserved
20	RCS0	x	ROM Location—Read only. This bit indicates the state of the ROM location (RCS0) configuration signal during reset. 0 ROM is located on PCI bus 1 ROM is located on processor/memory data bus
19	—	0	Reserved
18–17	PROC_TYPE	10	Processor type—These bits identify the type of processor used in the system. 10 603

Table 5-19. Bit Settings for PICR1—0xA8 (Continued)

Bit	Name	Reset Value	Description
16	ADDRESS_MAP	x	Address map—This bit controls which address map is used by the MPC8240. The initial state of this bit is determined by the inverse of the address map configuration signal (MAA0) during reset. Note that software that dynamically changes this bit must ensure that there are no pending PCI transactions and that there is a sync instruction following the address map change to allow the update to take effect. See Chapter 4, “Address Maps,” for more information. 0 The MPC8240 is configured for address map B. 1 The MPC8240 is configured for address map A (not supported when operating in PCI agent mode).
15–13	—	00	Reserved
12	FLASH_WR_EN	0	Flash write enable—This bit controls whether the MPC8240 allows write operations to Flash ROM. Note that if writes to Flash are enabled (with read-only devices in the banks), and a write transaction occurs, then bus contention may occur because the write data is driven on the data bus, and the read-only device starts driving the data bus. This can be avoided by disabling write capability to the Flash/ROM address space through the FLASH_WR_EN and/or FLASH_WR_LOCKOUT_EN configuration bits or by connecting the FOE signal to the output enable of the read-only device. 0 Flash write is disabled. 1 Flash write is enabled.
11	MCP_EN	0	Machine check enable—This bit controls whether the MPC8240 asserts MCP (and takes the machine check exception) upon detecting an error. See Chapter 13, “Error Handling and Exceptions,” for more information. 0 Machine check is disabled 1 Machine check is enabled
10	—	0	Reserved
9	CF_DPARK	0	Data bus park—This bit indicates whether the processor is parked on the data bus. 0 Processor is not parked on the data bus. 1 Processor is parked on the data bus. It is recommended that software set this bit.
8	—	0	Reserved
7	—	0	Reserved
6	ST_GATH_EN	0	This bit enables/disables store gathering of writes from the processor to PCI memory space. See Chapter 7, “Central Control Unit,” for more information. 0 Store gathering is disabled 1 Store gathering is enabled
5	LE_MODE	0	This bit controls the endian mode of the MPC8240. Note that this bit is also accessible from the external configuration register at 0x092. See Appendix B, “Bit and Byte Ordering,” for more information. 0 Big-endian mode 1 Little-endian mode

Table 5-19. Bit Settings for PICR1—0xA8 (Continued)

Bit	Name	Reset Value	Description
4	—	1	1 Reserved and must be set
3	CF_APARK	0	This bit indicates whether the processor address bus is parked. 0 Indicates that no processor is parked on the peripheral logic address bus 1 Indicates that the last processor that used the peripheral logic address bus is parked
2	Speculative PCI Reads	0	This bit controls speculative PCI reads from memory. Note that the peripheral logic block performs a speculative read in response to a PCI read-multiple command, even if this bit is cleared. See Chapter 7, "Central Control Unit," for more information. 0 Indicates that speculative reads are disabled. 1 Indicates that speculative reads are enabled.
1-0	—	00	Reserved

Figure 5-8 shows the bits of the PICR2.

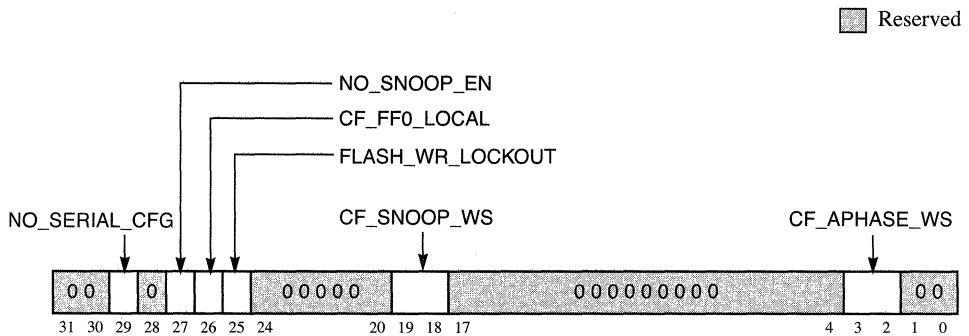


Figure 5-8. Processor Interface Configuration Register 2 (PICR2)—0xAC

Table 5-20 describes the bit settings for PICR2.

Table 5-20. Bit Settings for PICR2—0xAC

Bit	Name	Reset Value	Description
31-30	—	00	Reserved
29	NO_SERIAL_CFG	0	This bit controls whether the MPC8240 serializes configuration writes to PCI devices from the processor. 0 Configuration writes to PCI devices from the processor cause the MPC8240 to serialize and flush the internal buffers. 1 Configuration writes to PCI devices from the processor do not cause serialization. The internal buffers are not flushed.

Table 5-20. Bit Settings for PICR2—0xAC (Continued)

Bit	Name	Reset Value	Description
28	—	0	Reserved
27	NO_SNOOP_EN	0	This bit controls whether the MPC8240 generates snoop transactions on the peripheral bus for PCI-to-system memory transactions. This is provided as a performance enhancement for systems that do not need to maintain coherency on system memory accesses by PCI. 0 Snooping is enabled. 1 Snooping is disabled.
26	CF_FF0_LOCAL	0	ROM remapping enable—This bit allows the lower 8 Mbytes of the ROM/Flash address range to be remapped from the PCI bus to the processor/memory bus. Note that this bit is meaningful only if the ROM location parameter indicates that ROM is located on PCI bus (PICR1[RCS0] = 0). 0 ROM/Flash remapping disabled. The lower 8 Mbytes of the ROM/Flash address space are not remapped. All ROM/Flash accesses are directed to the PCI bus. 1 ROM/Flash remapping enabled. The lower 8 Mbytes of the ROM/Flash address space are remapped to the 60x/memory bus. ROM/Flash accesses in the range 0xFF00_0000–0xFF7F_FFFF are directed to the 60x/memory bus. ROM/Flash accesses in the range 0xFF80_0000–0xFFFF_FFFF are directed to the PCI bus.
25	FLASH_WR_LOCKOUT	0	Flash write lock-out—This bit, once set, prevents writing to Flash. Once set, this bit can only be cleared by a hard reset. 0 Write operations to Flash are enabled, provided FLASH_WR_EN = 1. 1 Write operations to Flash are disabled until the MPC8240 is reset.
24–20	—	0_0000	Reserved
19–18	CF_SNOOP_WS	11	Snoop wait states—These bits control the minimum number of wait states for the address phase in a snoop cycle. 00 0 wait states (2-clock address phase) 01 1 wait state (3-clock address phase) 10 2 wait states (4-clock address phase) 11 3 wait states (5-clock address phase) If the clock frequency ratio between the processor core and the memory interface is 1:1, this value can be changed to 0b10 for increased performance.
17–4	—	All 0s	Reserved

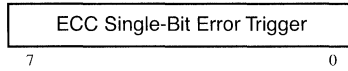


Figure 5-10. ECC Single-Bit Error Trigger Register—0xB9

Table 5-22 describes the bits of the ECC single-bit error trigger.

Table 5-22. Bit Settings for ECC Single-Bit Error Trigger Register—0xB9

Bit	Name	Reset Value	Description
7-0	ECC single-bit error trigger	All 0s	These bits provide the threshold value for the number of ECC single-bit errors that are detected before reporting an error condition. If the value of the single bit error counter register equals the value of this register, then an error is reported (provided ErrEnR1[2] = 1). If this register = 0x00, then no single bit error is ever generated.

5.7.2 Error Enabling Registers

Error enabling registers 1 and 2 (ErrEnR1 and ErrEnR2), shown in Figure 5-11 and Figure 5-12, control whether the MPC8240 recognizes and reports specific error conditions. Table 5-23 describes the bits of ErrEnR1; and Table 5-24 describes the bits of ErrEnR2.

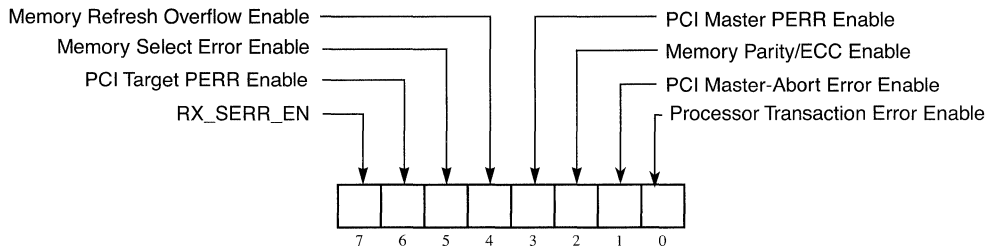


Figure 5-11. Error Enabling Register 1 (ErrEnR1)—0xC0

Table 5-23. Bit Settings for Error Enabling Register 1 (ErrEnR1)—0xC0

Bit	Name	Reset Value	Description
7	RX_SERR_EN	0	This bit enables the reporting of system errors that occur on the PCI bus (via SERR) for transactions involving the MPC8240 as a master. 0 Received PCI SERR disabled 1 Received PCI SERR enabled
6	PCI target PERR enable	0	This bit enables the reporting of data parity errors on the PCI bus for transactions involving the MPC8240 as a target. 0 Target PERR disabled 1 Target PERR enabled
5	Memory select error enable	0	This bit enables the reporting of memory select errors that occur on (attempted) accesses to system memory. 0 Memory select error disabled 1 Memory select error enabled
4	Memory refresh overflow enable	0	This bit enables the reporting of memory refresh overflow errors. 0 Memory refresh overflow disabled 1 Memory refresh overflow enabled
3	PCI master PERR enable	0	This bit enables the reporting of data parity errors on the PCI bus for transactions involving the MPC8240 as a master. 0 Master PERR disabled 1 Master PERR enabled
2	Memory parity/ECC enable	0	This bit enables the reporting of system memory read parity errors that occur on accesses to system memory or exceeding the ECC single-bit error threshold. [For SDRAM with inline ECC/parity, this is the memory write parity enable bit.] 0 Memory read parity/ECC single-bit threshold disabled 1 Memory read parity/ECC single-bit threshold enabled
1	PCI master-abort error enable	0	This bit enables the reporting of master-abort errors that occurred on the PCI bus for transactions involving the MPC8240 as a master. 0 PCI master-abort error disabled 1 PCI master-abort error enabled
0	Processor transaction error enable	1	This bit enables the reporting of processor transaction errors. 0 Processor internal bus error disabled 1 Processor internal bus error enabled

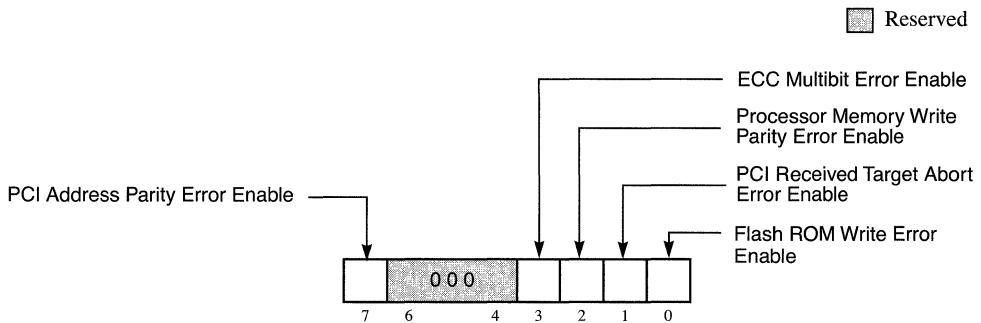


Figure 5-12. Error Enabling Register 2 (ErrEnR2)—0xC4

Table 5-24. Bit Settings for Error Enabling Register 2 (ErrEnR2)—0xC4

Bit	Name	Reset Value	Description
7	PCI address parity error enable	0	This bit controls whether the MPC8240 asserts MCP (provided MCP is enabled) if an address parity error is detected by the MPC8240 when acting as a PCI target. 0 PCI address parity errors disabled 1 PCI address parity errors enabled
6–4	—	000	Reserved
3	ECC multi-bit error enable	0	This bit enables the detection of ECC multibit errors. 0 ECC multi-bit error detection disabled 1 ECC multi-bit error detection enabled
2	Processor memory write parity error enable	0	This bit enables the detection of processor memory write parity errors (note: only applies for DSRAM with inline parity checking). 0 Processor memory write error detection disabled 1 Processor memory write error detection enabled
1	PCI received target abort error enable	0	This bit enables the detection of target abort errors received by the PCI interface. 0 Target abort error detection disabled 1 Target abort error detection enabled
0	Flash ROM write error enable	0	This bit controls whether the MPC8240 detects attempts to write to Flash when either PICR1[FLASH_WR_EN] = 0 or PICR2[FLASH_WR_LOCKOUT] = 0. 0 Disabled 1 Enabled

5.7.3 Error Detection Registers

Error detection registers 1 and 2 (ErrDR1 and ErrDR2), shown in Figure 5-13 and Figure 5-14, contain error flags that report when the MPC8240 detects a specific error condition.

The error detection registers are bit-reset type registers; that is, reading from these registers occurs normally; however, write operations are different in that bits (error flags) can be cleared but not set. A bit is cleared whenever the register is written, and the data in the corresponding bit location is a 1. For example, to clear bit 6 and not affect other bits in the register, write 0b0100_0000 to the register. When the MPC8240 detects an error, the appropriate error flag is set. Subsequent errors set the appropriate error flags in the error detection registers, but the bus error status and error address are not recorded until the previous error flags are cleared.

Error Handling Registers

Table 5-25 describes the bits of error detection register 1.

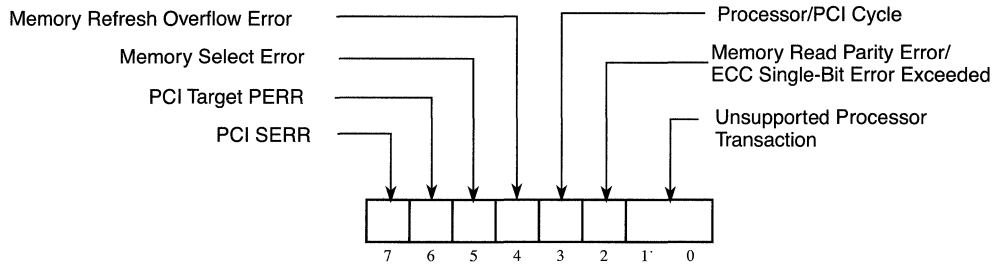


Figure 5-13. Error Detection Register 1 (ErrDR1)—0xC1

Table 5-26 describes the bits of error detection register 2.

Table 5-25. Bit Settings for Error Detection Register 1 (ErrDR1)—0xC1

Bit	Name	Reset Value	Description
7	PCI $\overline{\text{SERR}}$	0	PCI $\overline{\text{SERR}}$ 0 No system error signaled 1 System error signaled
6	PCI target $\overline{\text{PERR}}$	0	PCI target $\overline{\text{PERR}}$ 0 The MPC8240, as a PCI target, has not detected a data parity error 1 The MPC8240, as a PCI target, detected a data parity error
5	Memory select error	0	Memory select error 0 No error detected 1 Memory select error detected
4	Memory refresh overflow error	0	Memory refresh overflow error 0 No error detected 1 Memory refresh overflow has occurred
3	Processor/PCI cycle	0	Processor/PCI cycle 0 Error occurred on a processor-initiated cycle 1 Error occurred on a PCI-initiated cycle
2	Memory read parity error/ECC single-bit error trigger exceeded	0	Memory read parity error/ECC single-bit error trigger exceeded 0 No error detected 1 Parity error detected or ECC single-bit error trigger exceeded
1–0	Unsupported processor transaction	00	Unsupported processor transaction 00 No error detected 01 Unsupported transfer attributes. Refer to Chapter 13, “Error Handling and Exceptions”, for more details 10 Reserved 11 Reserved

 Reserved

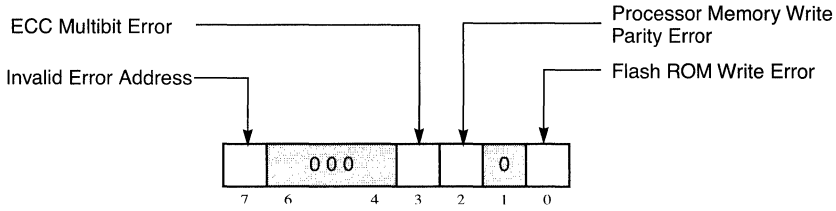


Figure 5-14. Error Detection Register 2 (ErrDR2)—0xC5

Table 5-26. Bit Settings for Error Detection Register 2 (ErrDR2)—0xC5

Bit	Name	Reset Value	Description
7	Invalid error address	0	This bit indicates whether the address stored in the processor/PCI error address register is valid. 0 The address in the error address register is valid. 1 The address in the error address register is not valid.
6–4	—	000	This bit is reserved.
3	ECC multi-bit error	0	ECC multibit error 0 No error detected 1 ECC multibit error detected
2	Processor memory write parity error	0	Processor memory write parity error (SDRAM with inline parity checking only). 0 No error detected 1 Processor memory write parity error detected
1	—	0	Reserved.
0	Flash ROM write error	0	Flash ROM write error 0 No error detected 1 The MPC8240 detected a write to Flash ROM when writes to Flash ROM are disabled.

5.7.4 Error Status Registers

The error status registers latch the state of the processor or PCI address bus when an error is detected. Figure 5-15, Figure 5-16, and Figure 5-17 show the system status registers used by error handling software.

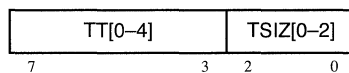


Figure 5-15. Internal Processor Bus Error Status Register—0xC3

Error Handling Registers

Table 5-27 describes the bits of the processor bus error status register.

Table 5-27. Bit Settings for Internal Processor Bus Error Status Register—0xC3

Bit	Name	Reset Value	Description
7–3	TT[0–4]	00000	These bits maintain a copy of TT[0–4]. When a processor bus error is detected, these bits are latched until all error flags are cleared.
2–0	TSIZ[0–2]	000	These bits maintain a copy of TSIZ[0–2]. When a processor bus error is detected, these bits are latched until all error flags are cleared.

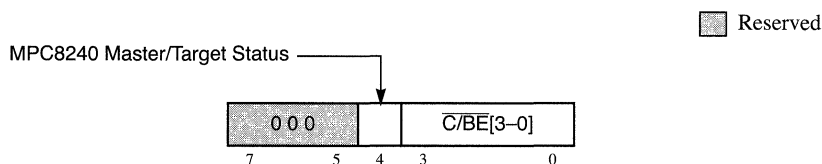


Figure 5-16. PCI Bus Error Status Register—0xC7

Table 5-28 describes the bits of the PCI bus error status register.

Table 5-28. Bit Settings for PCI Bus Error Status Register—0xC7

Bit	Name	Reset Value	Description
7–5	—	000	Reserved.
4	MPC8240 master/ target status	0	MPC8240 master/target status 0 MPC8240 is the PCI master 1 MPC8240 is the PCI target
3–0	$\overline{C}/\overline{BE}[3-0]$	0000	These bits maintain a copy of $\overline{C}/\overline{BE}[3-0]$. When a PCI bus error is detected, these bits are latched until all error flags are cleared.

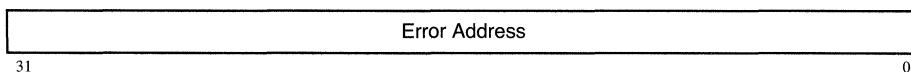


Figure 5-17. Processor/PCI Error Address Register—0xC8

Table 5-29 describes the bits of processor/PCI error address register.

Table 5-29. Bit Settings for Processor/PCI Error Address Register—0xC8

Bit	Name	Reset Value	Description
31–24	Error address	0x00	A[24–31] or AD[7–0]—dependent upon whether the error is a processor bus error or a PCI bus error. When an error is detected, these bits are latched until all error flags are cleared.
23–16		0x00	A[16–23] or AD[15–8]—(dependent upon whether the error is a processor bus error or a PCI bus error. When an error is detected, these bits are latched until all error flags are cleared.
15–8		0x00	A[8–15] or AD[23–16]—dependent upon whether the error is a processor bus error or a PCI bus error. When an error is detected, these bits are latched until all error flags are cleared.
7–0		0x00	A[0–7] or AD[31–24]—dependent upon whether the error is a processor bus error or a PCI bus error. When an error is detected, these bits are latched until all error flags are cleared.

5.8 Address Map B Options Register

The address map B options register (AMBOR) controls various configuration settings that can be used to alias some addresses and to control accesses to holes in the address map. Unrelated to address map B, there is also a bit that controls the operation of the DLL. See Section 3.3.2, “DLL Operation and Locking,” for more information about operation of the DLL.

Figure 5-18 shows the bits of the AMBOR.

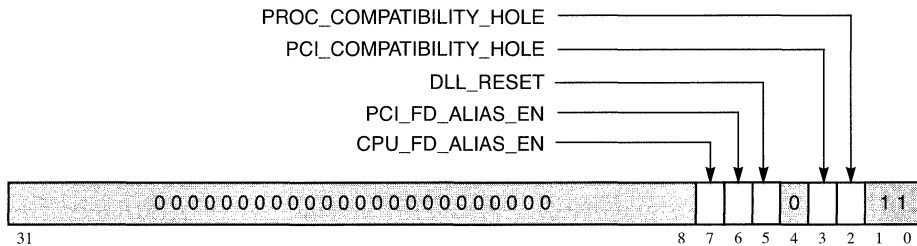


Figure 5-18. Address Map B Options Register (AMBOR)—0xE0

Table 5-30 shows the specific bit settings for the AMBOR.

Table 5-30. Bit Settings for the AMBOR—0xE0

Bits	Name	Reset Value	Description
31–8	—	All 0s	Reserved
7	CPU_FD_ALIAS_EN	1	Used to direct processor accesses to addresses that begin with 0xFDxx_xxxx. This bit is used only for address map B (and not supported in agent mode). 0 Access are routed normally 1 Processor accesses with 0xFDxx_xxxx address are forwarded to the PCI bus as PCI memory accesses to 0x00xx_xxxx.
6	PCI_FD_ALIAS_EN	1	Used to direct processor responses to addresses that begin with 0xFDxx_xxxx. This bit is used only for address map B (and not supported in agent mode). 0 No response 1 The MPC8240, as a PCI target, responds to addresses in the range 0xFD00_0000–0xFDFF_FFFF (asserts \overline{DEVSEL}), and forwards the transaction to system memory as 0x0000_0000–0x00FF_FFFF.
5	DLL_RESET	0	Used to reset the DLL tap point. See Section 3.3.2, “DLL Operation and Locking.” 0 DLL tries to lock the phase between the SDRAM_SYNC_IN signal and the internal sys_logic_clk signal. 1 The SDRAM_CLK signals are driven from tap point 0 of the internal delay line.
4	—	0	Reserved
3	PCI_COMPATIBILITY_HOLE	0	This bit is used only for address map B (and not supported in agent mode). 0 The MPC8240, as a PCI target, responds to PCI addresses in the range 0x000A_0000–0x000F_FFFF, and forwards the transaction to system memory. 1 The MPC8240, as a PCI target, does not respond to PCI addresses in the range 0x000A_0000–0x000F_FFFF.
2	PROC_COMPATIBILITY_HOLE	0	This bit is used only for address map B (and not supported in agent mode). 0 The MPC8240 forwards processor-initiated transactions in the address range 0x000A_0000–0x000B_FFFF to system memory. 1 The MPC8240 forwards processor-initiated transactions in the address range 0x000A_0000–0x000B_FFFF to the PCI memory space.
1–0	—	11	Reserved

5.9 Memory Interface Configuration Registers

The memory interface configuration registers (MICRs) control memory boundaries (starting and ending addresses), memory bank enables, memory timing, and external memory buffers. Initialization software must program the MICRs at reset and then enable the memory interface on the MPC8240 by setting the MEMGO bit in memory control configuration register 1 (MCCR1).

5.9.1 Memory Boundary Registers

The extended starting address and the starting address registers are used to define the lower address boundary for each memory bank. The lower boundary is determined by the following formula:

Lower boundary for bank $n = 0b00 \parallel \langle \text{extended starting address } n \rangle \parallel \langle \text{starting address } n \rangle \parallel 0x00000$.

The extended ending address and the ending address registers are used to define the upper address boundary for each memory bank. The upper boundary is determined by the following formula:

Upper boundary for bank $n = 0b00 \parallel \langle \text{extended ending address } n \rangle \parallel \langle \text{ending address } n \rangle \parallel 0xFFFFF$.

See Figure 5-19, Figure 5-20, and Table 5-31 for memory starting address register 1 and 2 bit settings.

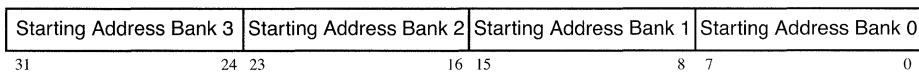


Figure 5-19. Memory Starting Address Register 1—0x80

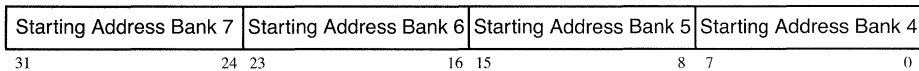


Figure 5-20. Memory Starting Address Register 2—0x84

Table 5-31. Bit Settings for Memory Starting Address Registers 1 and 2

Bit	Name	Reset Value	Description	Word Address
31–24	Starting address bank 3	0x00	Starting address for bank 3	0x80
23–16	Starting address bank 2	0x00	Starting address for bank 2	
15–8	Starting address bank 1	0x00	Starting address for bank 1	
7–0	Starting address bank 0	0x00	Starting address for bank 0	
31–24	Starting address bank 7	0x00	Starting address for bank 7	0x84
23–16	Starting address bank 6	0x00	Starting address for bank 6	
15–8	Starting address bank 5	0x00	Starting address for bank 5	
7–0	Starting address bank 4	0x00	Starting address for bank 4	

Memory Interface Configuration Registers

See Figure 5-21, Figure 5-22, and Table 5-32 for extended memory starting address register 1 and 2 bit settings.

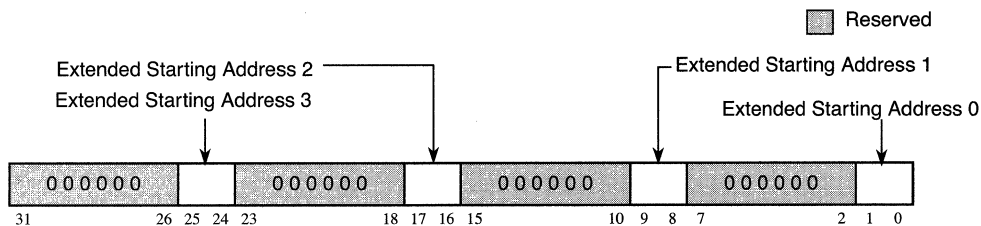


Figure 5-21. Extended Memory Starting Address Register 1—0x88.

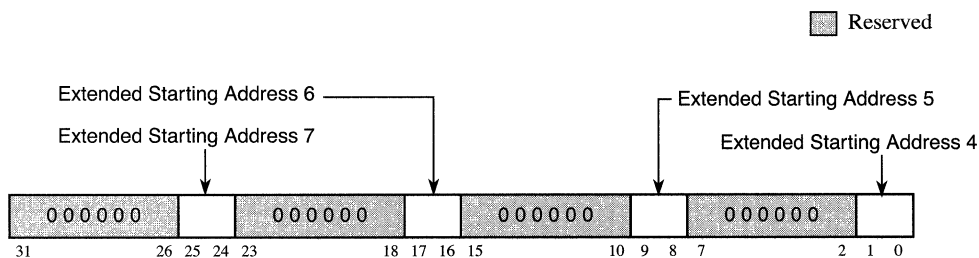


Figure 5-22. Extended Memory Starting Address Register 2—0x8C

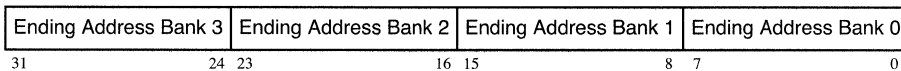
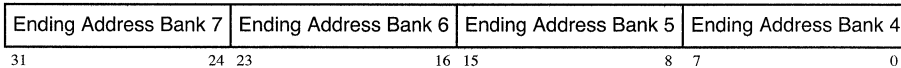
Table 5-32. Bit Settings for Extended Memory Starting Address Registers 1 and 2

Bit	Name	Reset Value	Description	Byte Address
31–26	—	All 0s	These bits are reserved.	0x88
25–24	Extended starting address 3	0b00	Extended starting address for bank 3	
23–18	—	All 0s	These bits are reserved.	
17–16	Extended starting address 2	0b00	Extended starting address for bank 2	
15–10	—	All 0s	These bits are reserved.	
9–8	Extended starting address 1	0b00	Extended starting address for bank 1	
7–2	—	All 0s	These bits are reserved.	
1–0	Extended starting address 0	0b00	Extended starting address for bank 0	

Table 5-32. Bit Settings for Extended Memory Starting Address Registers 1 and 2

Bit	Name	Reset Value	Description	Byte Address
31–26	—	All 0s	These bits are reserved.	0x8C
25–24	Extended starting address 7	0b00	Extended starting address for bank 7	
23–18	—	All 0s	These bits are reserved.	
17–16	Extended starting address 6	0b00	Extended starting address for bank 6	
15–10	—	All 0s	These bits are reserved.	
9–8	Extended starting address 5	0b00	Extended starting address for bank 5	
7–2	—	All 0s	These bits are reserved.	
1–0	Extended starting address 4	0b00	Extended starting address for bank 4	

See Figure 5-23, Figure 5-24, and Table 5-33 for memory ending address register 1 and 2 bit settings.

**Figure 5-23. Memory Ending Address Register 1—0x90****Figure 5-24. Memory Ending Address Register 2—0x94****Table 5-33. Bit Settings for Memory Ending Address Registers 1 and 2**

Bit	Name	Reset Value	Description	Byte Address
31–24	Ending address bank 3	0x00	Ending address for bank 3	0x90
23–16	Ending address bank 2	0x00	Ending address for bank 2	
15–8	Ending address bank 1	0x00	Ending address for bank 1	
7–0	Ending address bank 0	0x00	Ending address for bank 0	
31–24	Ending address bank 7	0x00	Ending address for bank 7	0x94
23–16	Ending address bank 6	0x00	Ending address for bank 6	
15–8	Ending address bank 5	0x00	Ending address for bank 5	
7–0	Ending address bank 4	0x00	Ending address for bank 4	

Memory Interface Configuration Registers

See Figure 5-25, Figure 5-26, and Table 5-34 for extended memory ending address register 1 and 2 bit settings.

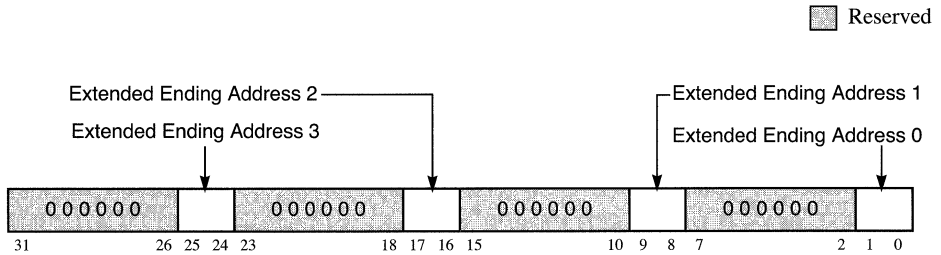


Figure 5-25. Extended Memory Ending Address Register 1—0x98

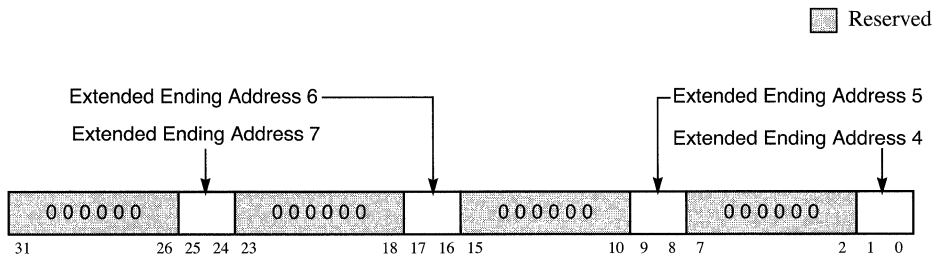


Figure 5-26. Extended Memory Ending Address Register 2—0x9C

Table 5-34. Bit Settings for Extended Memory Ending Address Registers 1 and 2

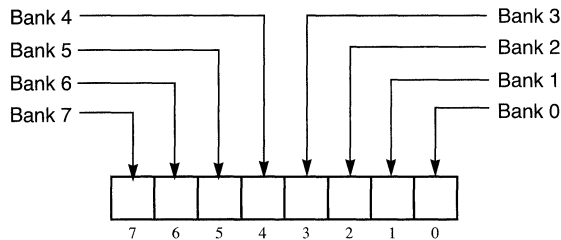
Bit	Name	Reset Value	Description	Byte Address
31–26	—	All 0s	These bits are reserved.	0x98
25–24	Extended ending address 3	0b00	Extended ending address for bank 3	
23–18	—	All 0s	These bits are reserved.	
17–16	Extended ending address 2	0b00	Extended ending address for bank 2	
15–10	—	All 0s	These bits are reserved.	
9–8	Extended ending address 1	0b00	Extended ending address for bank 1	
7–2	—	All 0s	These bits are reserved.	
1–0	Extended ending address 0	0b00	Extended ending address for bank 0	

Table 5-34. Bit Settings for Extended Memory Ending Address Registers 1 and 2 (Continued)

Bit	Name	Reset Value	Description	Byte Address
31–26	—	All 0s	These bits are reserved.	0x9C
25–24	Extended ending address 7	0b00	Extended ending address for bank 7	
23–18	—	All 0s	These bits are reserved.	
17–16	Extended ending address 6	0b00	Extended ending address for bank 6	
15–10	—	All 0s	These bits are reserved.	
9–8	Extended ending address 5	0b00	Extended ending address for bank 5	
7–2	—	All 0s	These bits are reserved.	
1–0	Extended ending address 4	0b00	Extended ending address for bank 4	

5.9.2 Memory Bank Enable Register

Individual banks are enabled or disabled by using the 1-byte memory bank enable register, shown in Figure 5-27 and Table 5-35. If a bank is enabled, the ending address of that bank must be greater than or equal to its starting address. If a bank is disabled, no memory transactions access that bank regardless of its starting and ending addresses.

**Figure 5-27. Memory Bank Enable Register—0xA0****Table 5-35. Bit Settings for Memory Bank Enable Register—0xA0**

Bit	Name	Reset Value	Description
7	Bank 7	0	Bank 7 0 Disabled 1 Enabled
6	Bank 6	0	Bank 6 0 Disabled 1 Enabled

Table 5-35. Bit Settings for Memory Bank Enable Register—0xA0 (Continued)

Bit	Name	Reset Value	Description
5	Bank 5	0	Bank 5 0 Disabled 1 Enabled
4	Bank 4	0	Bank 4 0 Disabled 1 Enabled
3	Bank 3	0	Bank 3 0 Disabled 1 Enabled
2	Bank 2	0	Bank 2 0 Disabled 1 Enabled
1	Bank 1	0	Bank 1 0 Disabled 1 Enabled
0	Bank 0	0	Bank 0 0 Disabled 1 Enabled

5.9.3 Memory Page Mode Register

The 1-byte memory page mode register, shown in Figure 5-28 and Table 5-36, contains the PGMAX parameter which controls how long the MPC8240 retains the currently accessed page (row) in memory. See Section 6.3.7, “FPM or EDO DRAM Page Mode Retention,” or Section 6.4.6, “SDRAM Page Mode Retention,” for more information.

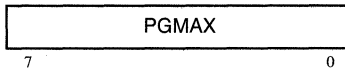


Figure 5-28. Memory Page Mode Register—0xA3

Table 5-36. Bit Settings for Memory Page Mode Register—0xA3

Bit	Name	Reset Value	Description
7–0	PGMAX	All 0s	For DRAM/EDO configurations, the value of PGMAX, multiplied by 64, determines the maximum RAS assertion interval for retained page mode. When programmed to 0x00, page mode is disabled. For SDRAM configurations, the value of PGMAX, multiplied by 64, determines the activate to precharge interval (sometimes called row active time or t_{RAS}) for retained page mode. When programmed to 0x00, page mode is disabled.

5.9.4 Memory Control Configuration Registers

The four 32-bit memory control configuration registers (MCCRs) set all RAM and ROM parameters. These registers are programmed by initialization software to adapt the MPC8240 to the specific memory organization used in the system. After all the memory configuration parameters have been properly configured, the initialization software turns on the memory interface using the MEMGO bit in MCCR1.

Note that the RAM_TYPE bit in MCCR1 must be cleared (to select SDRAM mode) before either the REGISTERED or INLINE buffer mode bits in MCCR4 are set to one. Inline or registered buffer modes are not supported in FPM/EDO DRAM systems and attempts to configure them may result in data corruption. This restriction includes the time between system reset and the setting of the MEMGO bit; therefore, it may dictate the order in which the memory controller configuration registers are written. It is recommended that the user first write MCCR1, 2, 3, and 4, in order, without setting the MEMGO bit. Afterwards, the user should perform a read-modify-write operation to set the MEMGO bit in MCCR1.

See Figure 5-29 and Table 5-37 for memory control configuration register 1 bit settings.

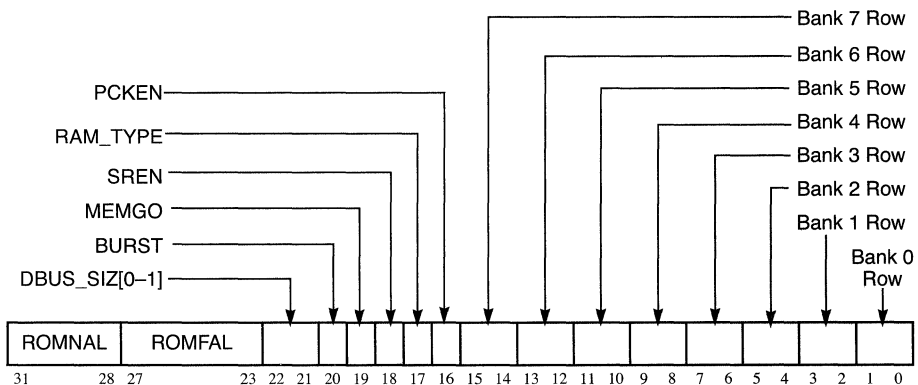


Figure 5-29. Memory Control Configuration Register 1 (MCCR1)—0xF0

Table 5-37. Bit Settings for MCCR1—0xF0

Bit	Name	Reset Value	Description
31–28	ROMNAL	All 1s	For burst-mode ROM and Flash reads, ROMNAL controls the next access time. The maximum value is 0b1111 (15). The actual cycle count is three cycles more than the binary value of ROMNAL. For Flash writes, ROMNAL measures the write pulse recovery (high) time. The maximum value is 0b1111 (15). The actual cycle count is four cycles more than the binary value of ROMNAL.
27–23	ROMFAL	All 1s	For nonburst ROM and Flash reads, ROMFAL controls the access time. For burst-mode ROMs, ROMFAL controls the first access time. The maximum value is 0b11111 (31). For the 64-bit and 32-bit configurations, the actual cycle count is three cycles more than the binary value of ROMFAL. For the 8-bit configuration, the actual cycle count is two cycles more than the binary value of ROMFAL. For Flash writes, ROMFAL measures the write pulse low time. The maximum value is 0b11111 (31). The actual cycle count is two cycles more than the binary value of ROMFAL.
22–21	DBUS_SIZ[0–1]	xx	Read-only—This field indicates the state of the ROM bank 0 data path width configuration signals [DL[0], FOE] at reset as follows. For ROM/Flash chip select #0 ($\overline{RCS0}$), 00 32-bit data bus. x1 8-bit data bus. 10 64-bit data bus. For ROM/Flash chip select #1 ($\overline{RCS1}$) and (S)DRAM, 0x 32-bit data bus. For FPM/EDO systems only, (RAM_TYPE=1). 1x 64-bit data bus.
20	BURST	0	Burst mode ROM timing enable 0 Indicates standard (nonburst) ROM access timing 1 Indicates burst-mode ROM access timing
19	MEMGO	0	RAM interface logic enable—Note that this bit must not be set until all other memory configuration parameters have been appropriately configured by boot code. 0 MPC8240 RAM interface logic disabled 1 MPC8240 RAM interface logic enabled
18	SREN	0	Self-refresh enable—Note that if self refresh is disabled, the system is responsible for preserving the integrity of DRAM/EDO/SDRAM during sleep mode. 0 Disables the DRAM/EDO/SDRAM self refresh during sleep mode 1 Enables the DRAM/EDO/SDRAM self refresh during sleep mode
17	RAM_TYPE	1	RAM type 0 Indicates synchronous DRAM (SDRAM) 1 Indicates DRAM or EDO DRAM (depending on the setting for MCCR2[EDO]) Note that this bit must be cleared (selecting DRAM or SDRAM) before the inline or registered buffer mode bits in MCCR4 are set.

Table 5-37. Bit Settings for MCCR1—0xF0 (Continued)

Bit	Name	Reset Value	Description
16	PCKEN	0	Memory interface parity checking/generation enable 0 Disables parity checking and parity generation for transactions to DRAM/EDO/SDRAM memory. If ECC is enabled, disables L2 parity checking. 1 Enables parity checking and generation for all memory transactions to DRAM/EDO/SDRAM. If ECC is enabled, enables L2 parity checking.
i5–i4	Bank 7 row	00	RAM bank 7 row address bit count—These bits indicate the number of row address bits that are required by the RAM devices in bank 7. For FPM/EDO DRAM configurations (RAM_TYPE = 1), the encoding is as follows: 00 9 row bits 01 10 row bits 10 11 row bits 11 12 or 13 row bits For SDRAM configurations (RAM_TYPE = 0), the encoding is as follows: 00 12 row bits by n column bits by 4 logical banks (12 × n × 4)—64 or 128 Mbit device 01 13 row bits by n column bits by 2 logical banks (13 × n × 2)—64 or 128 Mbit device 10 Reserved 11 11 row bits by n column bits by 2 logical banks (11 × n × 2)—16 Mbit device
13–12	Bank 6 row	00	RAM bank 6 row address bit count. See the description for Bank 7 row (bits 15–14).
11–10	Bank 5 row	00	RAM bank 5 row address bit count. See the description for Bank 7 row (bits 15–14).
9–8	Bank 4 row	00	RAM bank 4 row address bit count. See the description for Bank 7 row (bits 15–14).
7–6	Bank 3 row	00	RAM bank 3 row address bit count. See the description for Bank 7 row (bits 15–14).
5–4	Bank 2 row	00	RAM bank 2 row address bit count. See the description for Bank 7 row (bits 15–14).
3–2	Bank 1 row	00	RAM bank 1 row address bit count. See the description for Bank 7 row (bits 15–14).
1–0	Bank 0 row	00	RAM bank 0 row address bit count. See the description for Bank 7 row (bits 15–14).

Memory Interface Configuration Registers

See Figure 5-30 and Table 5-38 for memory control configuration register 2 (MCCR2) bit settings.

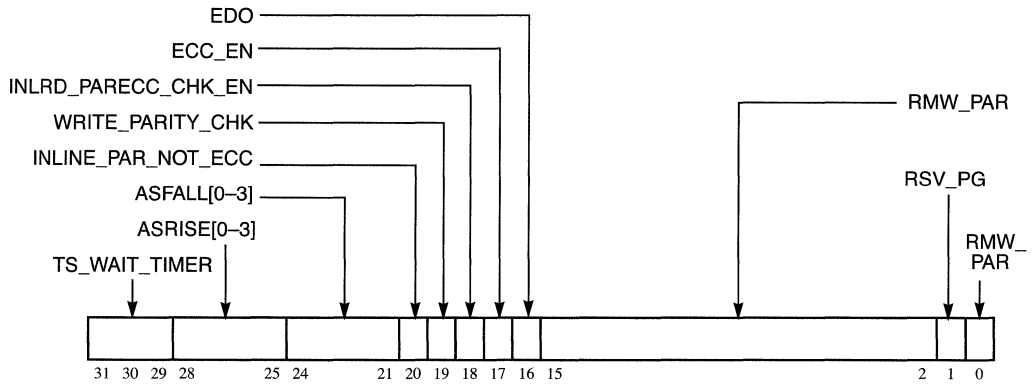


Figure 5-30. Memory Control Configuration Register 2 (MCCR2)—0xF4

Table 5-38. Bit Settings for MCCR2—0xF4

Bit	Name	Reset Value	Description
31–29	TS_WAIT_TIMER	000	<p>TS_WAIT_TIMER—These bits control the ROM output disable timing. The minimum time allowed for ROM devices to enter high impedance is 2 memory system clocks. TS_WAIT_TIMER adds (n-1) clocks to the minimum disable time. Negative additions (that is, setting TS_WAIT_TIMER[0..2] = 1) do not reduce the minimum enforced disable times. This delay is enforced after all ROM and Flash accesses preventing any other memory access from starting (for example, DRAM after ROM access, SDRAM after Flash access, ROM after Flash access).</p> <p>000 2 clocks minimum disable time 001 2 clocks minimum disable time 010 3 clocks minimum disable time 011 4 clocks minimum disable time 100 5 clocks minimum disable time 101 6 clocks minimum disable time 110 7 clocks minimum disable time 111 8 clocks minimum disable time</p>
28–25	ASRISE[0–3]	0000	<p>ASRISE[0–3]—These bits control the falling edge timing of the \overline{AS} signal for the Port X interface. See Section 6.5.6, “Port X Interface,” for more information.</p> <p>0000 Disables \overline{AS} signal generation 0001 1 clock 0010 2 clocks 0011 3 clocks ... 1111 15 clocks</p>

Table 5-38. Bit Settings for MCCR2—0xF4 (Continued)

Bit	Name	Reset Value	Description
24–21	ASFALL[0–3]	0000	<p>ASFALL[0–3]—These bits control the rising edge timing of the \overline{AS} signal for the Port X interface. See Section 6.5.6, “Port X Interface,” for more information.</p> <p>0000 0 clocks (\overline{AS} asserted coincident with the chip select) 0001 1 clock 0010 2 clocks 0011 3 clocks ... 1111 15 clocks</p>
20	INLINE_PAR_NOT_ECC	0	<p>Inline parity - not ECC—This bit selects between the ECC and parity checking/correction mechanisms of the inline data path when performing memory reads. This bit is applicable for SDRAM systems running in inline buffer mode (INLINE=1, bit 22 of MCCR4) ONLY.</p> <p>0 MPC8240 uses ECC on the memory data bus 1 MPC8240 uses parity on the memory data bus.</p>
19	WRITE_PARITY_CHK	0	<p>Write parity check enable—This bit controls whether the MPC8240 uses the parity checking hardware in the data path to report peripheral bus parity errors on memory system write operations. Write parity checking can be enabled for SDRAM, FPM, or EDO systems running in any buffer mode. This bit activates different parity checking hardware than that controlled by PCKEN.</p> <p>0 peripheral bus write parity error reporting disabled 1 peripheral bus write parity error reporting enabled</p>
18	INLRD_PARECC_CHK_EN	0	<p>Inline read parity or ECC check/correction enable. This bit controls whether the MPC8240 uses the ECC/parity checking and/or correction hardware in the inline data path to report ECC or parity errors on memory system read operations. Read parity/ECC checking can be enabled for SDRAM systems running in inline buffer mode (INLINE=1, bit 22 of MCCR4) only. This bit activates different parity/ECC checking/correction hardware than that controlled by ECC_EN and PCKEN.</p> <p>0 inline memory bus read parity/ECC error reporting disabled 1 inline memory bus read parity/ECC error reporting enabled</p>
17	ECC_EN	0	<p>ECC enable—This bit controls whether the MPC8240 uses ECC for transactions to system memory. ECC_EN should be set only for systems using FPM/EDO memory. Note that the ECC_EN parameter overrides the PCKEN parameter. Also note that this bit and RMW_PAR cannot both be set to, and it is illegal to set this bit with EDO = 1 and REGISTERED = 1. Systems using SDRAM use a different (inline) ECC hardware and therefore, must have ECC_EN = 0. See Section 6.3, “FPM or EDO DRAM Interface Operation,” for more information.</p> <p>When ECC_EN = 1, the processor should not be configured to check address or data parity in the HID0 register. (See Section 2.3.1.2.1, “Hardware Implementation-Dependent Register 0 (HID0).” 0 ECC disabled 1 ECC enabled</p>

Table 5-38. Bit Settings for MCCR2—0xF4 (Continued)

Bit	Name	Reset Value	Description
16	EDO	0	EDO enable—This bit indicates the type of DRAMs for the MPC8240 memory interface. See Section 6.3, “FPM or EDO DRAM Interface Operation,” for more information. 0 Indicates standard DRAMs 1 Indicates EDO DRAMs
15–2	REFINT	All 0s	Refresh interval—These bits directly represent the number of clock cycles between CBR refresh cycles. One row is refreshed in each RAM bank during each CBR refresh cycle. The value for REFINT depends on the specific RAMs used and the operating frequency of the MPC8240. See Section 6.3.10, “FPM or EDO DRAM Refresh,” or Section 6.4.13, “SDRAM Refresh,” for more information. Note that the period of the refresh interval must be greater than the read/write access time to insure that read/write operations complete successfully.
1	RSV_PG	0	RSV_PG— If this bit is set, the MPC8240 reserves one of the four page registers at all times. This is equivalent to only allowing three simultaneous open pages. 0 Four open page mode (default) 1 Reserve one of the four page registers at all times
0	RMW_PAR	0	Read-modify-write (RMW) parity enable— This bit controls how the MPC8240 writes parity bits to DRAM/EDO/SDRAM. Note that this bit does not enable parity checking and generation. PCKEN must be set to enable parity checking. Also note that this bit and ECC_EN cannot both be set to 1. See Section 6.3.8, “FPM or EDO DRAM Parity and RMW Parity,” and Section 6.4.10, “SDRAM Parity and RMW Parity,” for more information. 0 RMW parity disabled 1 RMW parity enabled

See Figure 5-31 and Table 5-39 for memory control configuration register 3 (MCCR3) bit settings.

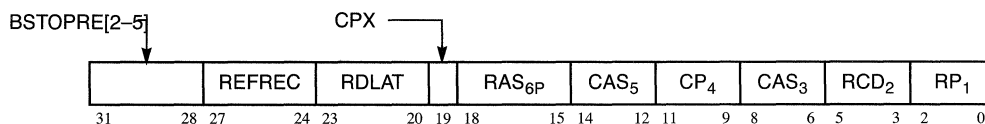


Figure 5-31. Memory Control Configuration Register 3 (MCCR3)—0xF8

Table 5-39. Bit Settings for MCCR3—0xF8

Bit	Name	Reset Value	Description
31–28	BSTOPRE[2–5]	0000	Burst to precharge—bits 2–5—For SDRAM only. These bits, together with BSTOPRE[0–1] (bits 19–18 of MCCR4), and BSTOPRE[6–9] (bits 3–0 of MCCR4), control the open page interval. The page open duration counter is reloaded with BSTOPRE[0–9] every time the page is accessed (including page hits). When the counter expires, the open page is closed with a SDRAM-precharge bank command.
27–24	REFREC	0000	Refresh to activate interval—For SDRAM only. These bits control the number of clock cycles from an SDRAM-refresh command until an SDRAM-activate command is allowed. See Section 6.4.13, “SDRAM Refresh,” for more information. 0001 1 clock 0010 2 clocks 0011 3 clocks 1111 15 clocks 0000 16 clocks
23–20	RDLAT	0000	Data latency from read command—For SDRAM only. These bits control the number of clock cycles from an SDRAM-read command until the first data beat is available on the data bus. RDLAT values greater than 6 clocks are not supported. See Section 6.4.7, “SDRAM Power on Initialization,” for more information. Note that for SDRAM, this value must be programmed to a valid value (from the reset value). 0000 Reserved 0001 1 clock 0010 2 clocks 0011 3 clocks 0100 4 clocks 0101 5 clocks 0110 6 clocks 0111 reserved (not supported) 1111 reserved (not supported)
19	CPX	0	$\overline{\text{CAS}}$ write timing modifier—For DRAM/EDO only. This bit, when set, adds one clock cycle to the $\overline{\text{CAS}}$ precharge interval ($\text{CP}_4 + 1$) and subtracts one clock cycle from the $\overline{\text{CAS}}$ assertion interval for page mode access ($\text{CAS}_5 - 1$) for write operations to DRAM/EDO. Read operations are unmodified. See Section 6.3.5, “FPM or EDO DRAM Interface Timing,” for more information. 0 $\overline{\text{CAS}}$ write timing is unmodified 1 $\overline{\text{CAS}}$ write timing is modified as described above

Table 5-39. Bit Settings for MCCR3—0xF8 (Continued)

Bit	Name	Reset Value	Description
18–15	RAS _{6P}	0000	<p>$\overline{\text{RAS}}$ assertion interval for CBR refresh—For DRAM/EDO only. These bits control the number of clock cycles $\overline{\text{RAS}}$ is held asserted during CBR refresh. The value for RAS_{6P} depends on the specific DRAMs used and the frequency of the memory interface. See Section 6.3.10, “FPM or EDO DRAM Refresh,” for more information.</p> <p>0001 1 clock 0010 2 clocks 0011 3 clocks 1111 15 clocks 0000 16 clocks</p>
14–12	CAS ₅	000	<p>$\overline{\text{CAS}}$ assertion interval for page mode access—For DRAM/EDO only. These bits control the number of clock cycles $\overline{\text{CAS}}$ is held asserted during page mode accesses. The value for CAS₅ depends on the specific DRAMs used and the frequency of the memory interface. Note that when ECC is enabled, CAS₅ + CP₄ must equal four clock cycles. See Section 6.3.5, “FPM or EDO DRAM Interface Timing,” for more information.</p> <p>001 1 clock 010 2 clocks 011 3 clocks 111 7 clocks 000 8 clocks</p>
11–9	CP ₄	000	<p>$\overline{\text{CAS}}$ precharge interval—For DRAM/EDO only. These bits control the number of clock cycles that $\overline{\text{CAS}}$ must be held negated in page mode (to allow for column precharge) before the next assertion of $\overline{\text{CAS}}$. Note that when ECC is enabled, CAS₅ + CP₄ must equal four clock cycles. See Section 6.3.5, “FPM or EDO DRAM Interface Timing,” for more information.</p> <p>001 1 clock 010 2 clocks 011 reserved 111 reserved 000 reserved</p>
8–6	CAS ₃	000	<p>$\overline{\text{CAS}}$ assertion interval for the first access—For DRAM/EDO only. These bits control the number of clock cycles $\overline{\text{CAS}}$ is held asserted during a single beat or during the first access in a burst. The value for CAS₃ depends on the specific DRAMs used and the frequency of the memory interface. See Section 6.3.5, “FPM or EDO DRAM Interface Timing,” for more information.</p> <p>001 1 clock 010 2 clocks 011 3 clocks 111 7 clocks 000 8 clocks</p>

Table 5-39. Bit Settings for MCCR3—0xF8 (Continued)

Bit	Name	Reset Value	Description
5–3	RCD ₂	000	<p>RAS to CAS delay interval—For DRAM/EDO only. These bits control the number of clock cycles between the assertion of RAS and the first assertion of CAS. The value for RCD₂ depends on the specific DRAMs used and the frequency of the memory interface. However, RCD₂ must be at least two clock cycles. See Section 6.3.5, “FPM or EDO DRAM Interface Timing,” for more information.</p> <p>001 Reserved 010 2 clocks 011 3 clocks 111 7 clocks 000 8 clocks</p>
2–0	RP ₁	000	<p>RAS precharge interval—For DRAM/EDO only. These bits control the number of clock cycles that RAS must be held negated (to allow for row precharge) before the next assertion of RAS. Note that RP₁ must be at least two clock cycles and no greater than 5 clock cycles. See Section 6.3.5, “FPM or EDO DRAM Interface Timing,” for more information.</p> <p>010 2 clocks 011 3 clocks 110 4 clocks 101 5 clocks all others: Reserved</p>

See Figure 5-32 and Table 5-40 for memory control configuration register 4 (MCCR4) bit settings.

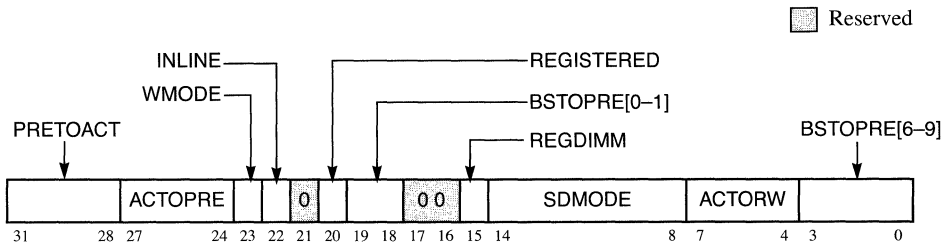


Figure 5-32. Memory Control Configuration Register 4 (MCCR4)—0xFC

Table 5-40. Bit Settings for MCCR4—0xFC

Bit	Name	Reset Value	Description
31–28	PRETOACT	0000	Precharge to activate interval—For SDRAM only. These bits control the number of clock cycles from an SDRAM-precharge command until an SDRAM-activate command is allowed. See Section 6.4.7, “SDRAM Power on Initialization,” for more information. 0001 1 clock 0010 2 clocks 0011 3 clocks 1111 15 clocks 0000 16 clocks
27–24	ACTOPRE	0000	Activate to precharge interval—For SDRAM only. These bits control the number of clock cycles from an SDRAM-activate command until an SDRAM-precharge command is allowed. See Section 6.4.7, “SDRAM Power on Initialization,” for more information. 0001 1 clock 0010 2 clocks 0011 3 clocks 1111 15 clocks 0000 16 clocks
23	WMODE	0	Length of burst for 32-bit data.—Applies to 32-bit data path mode only. Determines whether the burst ROMs can accept eight beats in a burst or only four. In 32-bit data path mode, burst transactions require data beats. If the burst ROM can only accept four beats per burst, the memory controller must perform two transactions to the ROM. 0 Four beats per burst (default) 1 Eight beats per burst
22	INLINE	0	Enable the inline ECC and parity check hardware—For SDRAM only; must be cleared for FPM/EDO DRAM. The inline ECC and parity hardware allows MPC8240 to check/generate parity on the internal peripheral bus and check/correct/generate ECC or parity on the external SDRAM memory bus. REGISTERED must be cleared when using this function. See Chapter 6, “MPC8240 Memory Interface,” for more information. 0 Inline ECC/parity disabled 1 Inline ECC/parity enabled
21	—	00	Reserved
20	REGISTERED	0	Memory data interface type—Must be cleared for FPM/EDO DRAM; must be cleared for SDRAM with inline ECC/Parity; see Chapter 6, “MPC8240 Memory Interface,” for more information. 0 Flow through or transparent latch type buffer unless INLINE = 1 1 Registered type buffer

Table 5-40. Bit Settings for MCCR4—0xFC (Continued)

Bit	Name	Reset Value	Description
19–18	BSTOPRE[0–1]	00	Burst to precharge—bits 0–1—For SDRAM only. These bits, together with BSTOPRE[2–5] (bits 31–28 of MCCR3), and BSTOPRE[6–9] (bits 3–0 of MCCR4), control the open page interval. The page open duration counter is reloaded with BSTOPRE[0–9] every time the page is accessed (including page hits). When the counter expires, the open page is closed with a SDRAM-precharge bank command. See Chapter 6, “MPC8240 Memory Interface,” for more information.
17–16	—	00	Reserved
15	REGDIMM	0	Registered DIMMs—Memory data and parity data path buses configured for registered DIMMs. For SDRAM only. When enabled (REGDIMM = 1), SDRAM write data and parity are delayed by one cycle on the memory bus with respect to the SDRAM control signals (for example, SDRAS, SDCAS, \overline{WE}). 0 Normal DIMMs 1 Registered DIMMs selected
14–8	SDMODE	All 0s	SDRAM mode register—For SDRAM only. These bits specify the SDRAM mode register data to be written to the SDRAM array during power-up configuration. Note that the SDRAM mode register “opcode” field is not specified and is forced to b'0_0000' by the MPC8240 when the mode registers are written. Bit Description 14–12 CAS latency 000 Reserved 001 1 010 2 011 3 100 Reserved 101 Reserved 110 Reserved 111 Reserved 11 Wrap type 0 Sequential. Default for MPC8240 1 Interleaved - Reserved 10–8 Burst length 000 Reserved 001 Reserved 010 4 011 8 100 Reserved 101 Reserved 110 Reserved 111 Reserved

Table 5-40. Bit Settings for MCCR4—0xFC (Continued)

Bit	Name	Reset Value	Description
7–4	ACTORW	0000	<p>Activate to read/write interval—For SDRAM only. These bits control the number of clock cycles from an SDRAM-activate command until an SDRAM-read or SDRAM-write command is allowed. See Section 6.4.7, “SDRAM Power on Initialization,” for more information.</p> <p>0001 Reserved</p> <p>0010 2 clocks (minimum for flow-through or registered data interfaces)</p> <p>0011 3 clocks (minimum for inline ECC/parity data interfaces)</p> <p>... ..</p> <p>1111 15 clocks</p> <p>0000 16 clocks</p>
3–0	BSTOPRE[6–9]	0000	<p>Burst to precharge—bits 6–9—For SDRAM only. These bits, together with BSTOPRE[0–1] (bits 19–18 of MCCR4), and BSTOPRE[2–5] (bits 31–28 of MCCR3), control the open page interval. The page open duration counter is reloaded with BSTOPRE[0–9] every time the page is accessed (including page hits). When the counter expires, the open page is closed with a SDRAM-precharge bank command. See Chapter 6, “MPC8240 Memory Interface,” for more information.</p>

Chapter 6

MPC8240 Memory Interface

The MPC8240 integrates a high-performance memory controller that controls processor and PCI interactions to local memory. The MPC8240 supports various types of DRAM and ROM/Flash configurations as local memory. All devices connected to the MPC8240's memory interface must be 3.3-V I/O compliant.

- DRAM—fast page mode (FPM) and extended data out (EDO)
 - High-bandwidth bus (32- or 64-bit data bus) to DRAM
 - One Mbyte to 1 Gbyte DRAM memory space
 - One to eight chip selects of 4-, 16-, 64- or 128-Mbit memory devices
 - Programmable timing for FPM and EDO
- SDRAM
 - SDRAMs must comply with the JEDEC specification for SDRAM
 - High-bandwidth bus (32- or 64-bit data bus) to SDRAM
 - One Mbyte to 1 Gbyte SDRAM memory— 1 to 8 chip selects for SDRAM bank sizes ranging from 1 Mbyte to 128 Mbytes per bank
 - Supports page mode SDRAMs—four open pages simultaneously
 - Programmable timing for SDRAMs
- ROM/Flash
 - 16 Mbytes of ROM/Flash space can be divided between the PCI bus and the memory bus (8 Mbytes each)
 - Supports 8-bit asynchronous ROM or 64-bit burst-mode ROM
 - Configurable data-path—8-, 32-, or 64-bit
 - Supports bus-width writes to Flash
- Port X—The ROM/Flash controller can interface any device that can be controlled with an address and data field (communication devices, DSPs, general purpose I/O devices, or registers). Some devices may require a small amount of external logic to properly generate address strobes and chip selects.
 - 8-bit Port X
 - 32-bit Port X
 - 64-bit Port X—the floating-point (FPU) unit must be enabled for 64-bit writes

- Data-path buffering—72-bits (64-bit data and 8-bit parity)
 - Reduces loading on the internal processor core bus
 - Reduces loading of the drivers of the memory system
 - Reduces signal trace delay known as time-of-flight (TOF)
- Parity—Supports normal parity and read-modify-write (RMW)
- Error checking and correction (ECC)—64-bit only
 - DRAM ECC—Located in the central control unit (CCU)
 - SDRAM ECC—Located in-line with the data-path buffers

The MPC8240 is designed to control a 32- or 64-bit data-path to main memory (SDRAM or DRAM). The MPC8240 can be configured to check parity or ECC on memory reads. Parity checking and generation can be enabled with 4-parity bits for a 32-bit data-path or 8-parity bits for 64-bit data-path. Concurrent ECC is only generated for 64-bit data-path with 8-syndrome bits.

The MPC8240 supports SDRAM or DRAM bank sizes from 1 to 128 Mbytes and provides bank start address and end address configuration registers. However the MPC8240 does not support mixed SDRAM or DRAM configurations.

Note

There are at least three types of banks:

1. Physical—Dual in-line memory modules (DIMMs)
2. Logical—Chip selects
3. Internal—Bank selects

The MPC8240 can be configured so that appropriate row and column address multiplexing will occur on a chip-select-by-chip-select basis. Addresses (DRAM or SDRAM) and bank selects (SDRAM only) are provided through a 14-bit interface for SDRAM and 13-bit interface for DRAM.

ROM/Flash systems are supported by up to 21 address bits, 2 bank selects, 1 write enable and 1 output enable. Figure 6-1 is a block diagram of the memory interface.

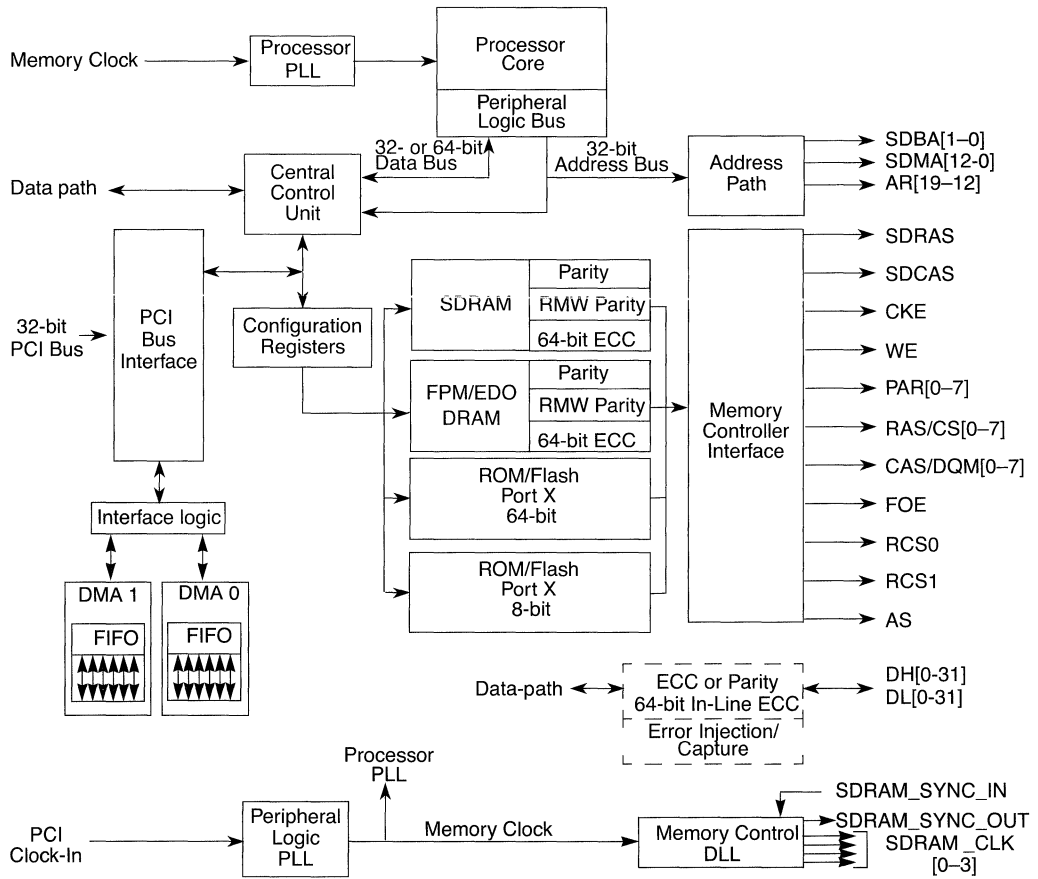


Figure 6-1. Block Diagram for Memory Interface

6.1 Memory Interface Signal Summary

Table 6-1 summarizes the memory interface signals.

Note

Some signals function differently depending on the type of memory system the MPC8240 is configured to support.

Table 6-1. Memory Interface Signal Summary

Signal Name	Signal Name	Alternate Function	Pins	I/O
$\overline{RAS}[0-7]$	DRAM row address strobe 0-7	$\overline{CS}[0-7]$	8	O
$\overline{CAS}[0-7]$	DRAM column address strobe 0-7	DQM[0-7]	8	O
$\overline{CS}[0-7]$	SDRAM chip select 0-7	$\overline{RAS}[0-7]$	8	O
DQM[0-7]	SDRAM data mask in/out 0-7	$\overline{CAS}[0-7]$	8	O
WE	Write enable	—	1	O
SDMA[12-0]	SDRAM address 12-0	See Table 6-2, "Memory Address Signal Mappings"	13	O
SDBA[1-0]	SDRAM bank select 1-0		2	O
DH[0-31]	Data bus high	—	32	I/O
DL[0-31] DL[0] ¹	Data bus low	—	32	I/O
PAR[0-7]	Data parity 0-7	AR[19-12]	8	I/O
AR[19-12]	ROM address 19-12	PAR[0-7]	8	O
CKE ¹	SDRAM clock enable	—	1	O
SDRAS	SDRAM row address strobe	—	1	O
SDCAS	SDRAM column address strobe	—	1	O
$\overline{RCS0}$ ¹	ROM or bank 0 select	—	1	O
$\overline{RCS1}$	ROM or bank 1select	—	1	O
\overline{FOE} ¹	Flash output enable	—	1	O
\overline{AS} ¹	Address strobe for Port X	—	1	O

¹The MPC8240 samples these signals at the negation of reset to determine the reset configuration. After they are sampled, they assume their normal functions. See Section 3.4, "Configuration Pins Sampled at Reset," for more information about their function during reset.

Table 6-2 shows memory address signal mappings.

Table 6-2. Memory Address Signal Mappings

MPC8240 Signal Name (Outputs)		Logical Names						JEDEC DIMM Signals (Inputs)	
		FPM/ EDO DRAM Address	16- Mbit SDRAM Address 2-bank	64- 128- Mbit SDRAM Address 2-bank	64- 128- Mbit SDRAM Address 4-bank	ROM/ Flash Address 8- and 32-bit mode	ROM/ Flash Address 64-bit mode	SDRAM 168-pin DIMM	FPM/ EDO 168-pin DIMM
msb	SDMA12/ SDBA1	MA12	–	SDMA12	SDBA1	AR20	–	BA1	A12
	PAR0					AR19	AR19		
	PAR1					AR18	AR18		
	PAR2					AR17	AR17		
	PAR3					AR16	AR16		
	PAR4					AR15	AR15		
	PAR5					AR14	AR14		
	PAR6					AR13	AR13		
	PAR7					AR12	AR12		
	SDBA0	MA11	SDBA0	SDBA0	SDBA0	AR11	AR11	BA0	A11
	SDMA11	–	–	SDMA11	SDMA11	–	–	A11	–
	SDMA10	MA10	SDMA10	SDMA10	SDMA10	AR10	AR10	A10(AP)	A10
	SDMA9	MA9	SDMA9	SDMA9	SDMA9	AR9	AR9	A9	A9
	SDMA8	MA8	SDMA8	SDMA8	SDMA8	AR8	AR8	A8	A8
	SDMA7	MA7	SDMA7	SDMA7	SDMA7	AR7	AR7	A7	A7
	SDMA6	MA6	SDMA6	SDMA6	SDMA6	AR6	AR6	A6	A6
	SDMA5	MA5	SDMA5	SDMA5	SDMA5	AR5	AR5	A5	A5
	SDMA4	MA4	SDMA4	SDMA4	SDMA4	AR4	AR4	A4	A4
	SDMA3	MA3	SDMA3	SDMA3	SDMA3	AR3	AR3	A3	A3
	SDMA2	MA2	SDMA2	SDMA2	SDMA2	AR2	AR2	A2	A2
SDMA1	MA1	SDMA1	SDMA1	SDMA1	AR1	AR1	A1	A1	
lsb	SDMA0	MA0	SDMA0	SDMA0	SDMA0	AR0	AR0	A0	A0

6.2 Memory Interface Configuration at Reset

Three pins are sampled at reset to determine the following memory bus information:

- \overline{FOE}
 - 1 = 8 bit Flash, ROM, or Port X data bus width configuration for bank 0
 - 0 = 32 or 64 bit Flash, ROM, or Port X data bus width configuration for bank 0
- $\overline{RCS0}$
 - 0 = ROM on PCI bus
 - 1 = ROM on local memory bus
- DL[0]
 - 0 = Memory interface is 32 bits—SDRAM, DRAM, ROM, Flash, or Port X
 - 1 = Memory interface is 64 bits—SDRAM, DRAM, ROM, Flash, or Port X

Refer to Section 3.4, “Signal Descriptions and Clocking,” for more information.

6.3 FPM or EDO DRAM Memory Interface Operation

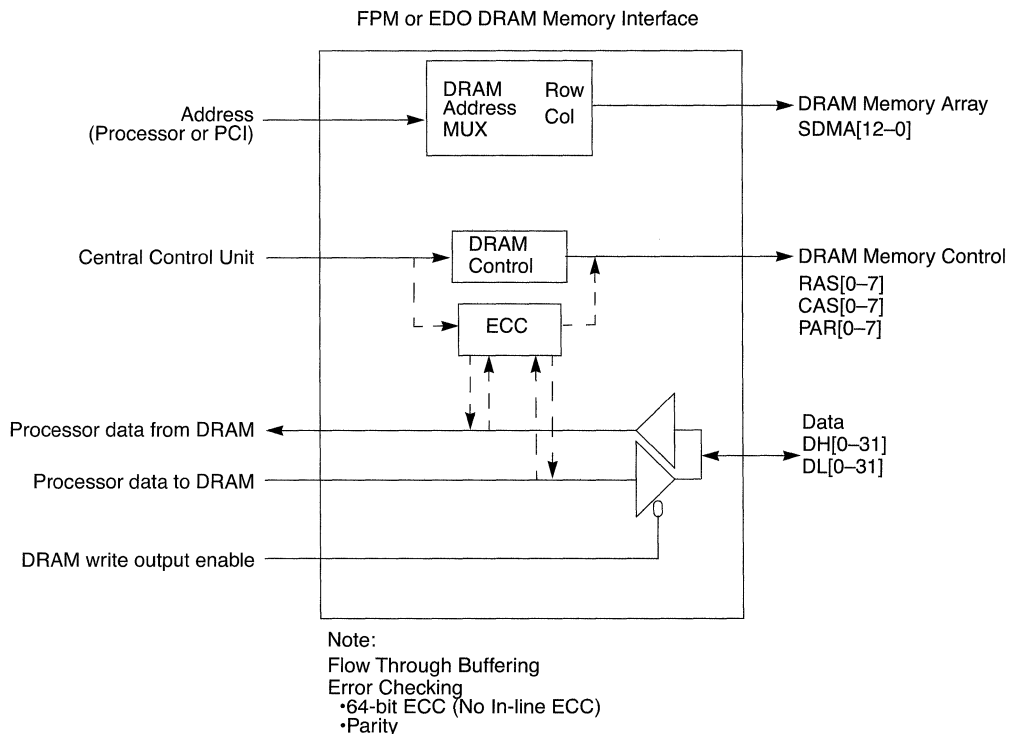


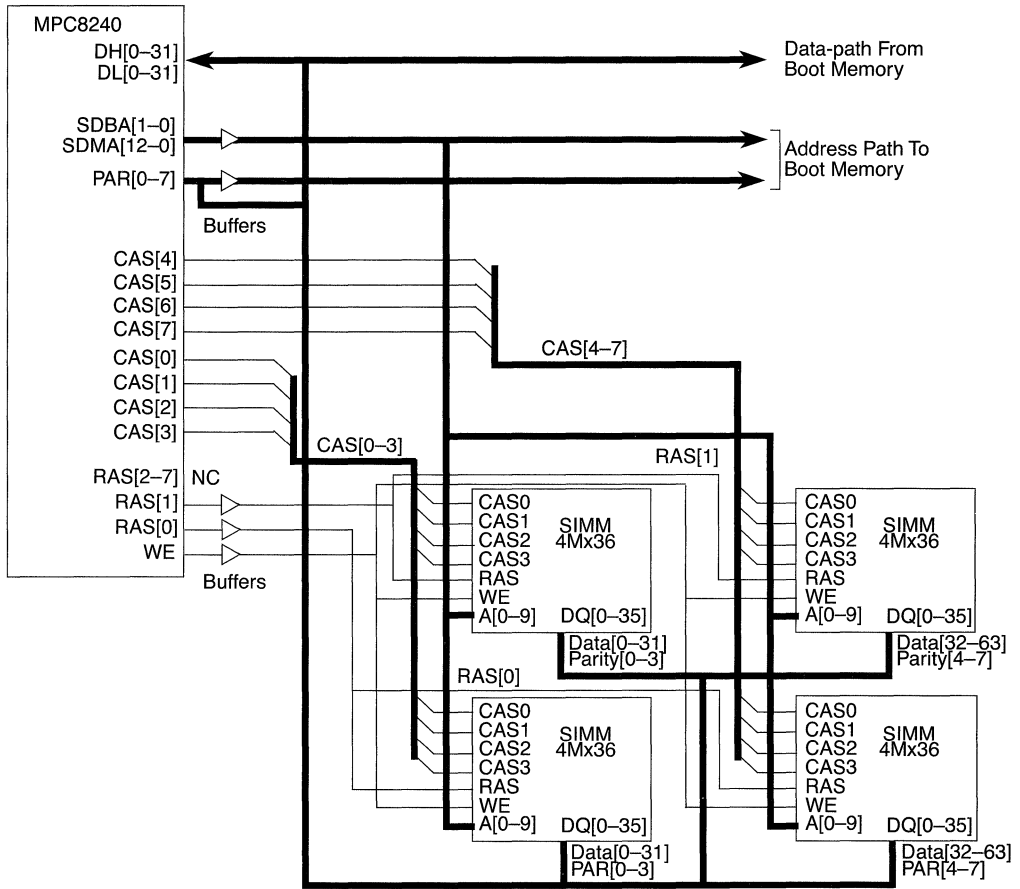
Figure 6-2. FPM or EDO DRAM Memory Interface Block Diagram

The MPC8240 supports a variety of DRAM configurations, through SIMM, DIMM, or direct board attachment. Thirteen address pins provide for DRAM device densities up to 64 Mbits. Eight row address strobe ($\overline{\text{RAS}}$) signals support up to eight banks of memory. Each bank can be 8 bytes wide; eight column address strobe ($\overline{\text{CAS}}$) signals are used to provide byte selection for writes. The banks can be built of DRAMs, SIMMs or DIMMs that range from 4 to 128 Mbits as described in Table 6-4. The memory design must be byte-selectable for writes using $\overline{\text{CAS}}$. The MPC8240 allows up to 1 Gbyte of addressable memory.

In addition to the $\overline{\text{CAS}}[0-7]$ signals, $\overline{\text{RAS}}[0-7]$ signals, and address signals $\text{SDMA}[0-12]$ and $\text{SDBA}[1-0]$, there are 64 data signals $\text{DH}[0-31]$ and $\text{DL}[0-31]$, a write enable ($\overline{\text{WE}}$) signal, and one parity bit per byte-width of data $\text{PAR}[0-7]$ for a total of 102 DRAM memory signals. Figure 6-3 is an example of a two-bank 16 Mbyte DRAM system.

Some address and control signals may or may not require buffering, depending upon the system design. Analysis of the MPC8240 AC specifications, desired memory operating frequency, capacitive loads, and board routing loads assist the system designer in deciding whether any signals require buffering. See the MPC8240 Hardware Specifications.

FPM or EDO DRAM Interface Operation



Address Signals (Outputs)	SDMA										
		9	8	7	6	5	4	3	2	1	0
Logical Names	msb		MA						lsb		
	9	8	7	6	5	4	3	2	1	0	

Figure 6-3. Example 16-Mbyte DRAM System with Parity—64-Bit Mode

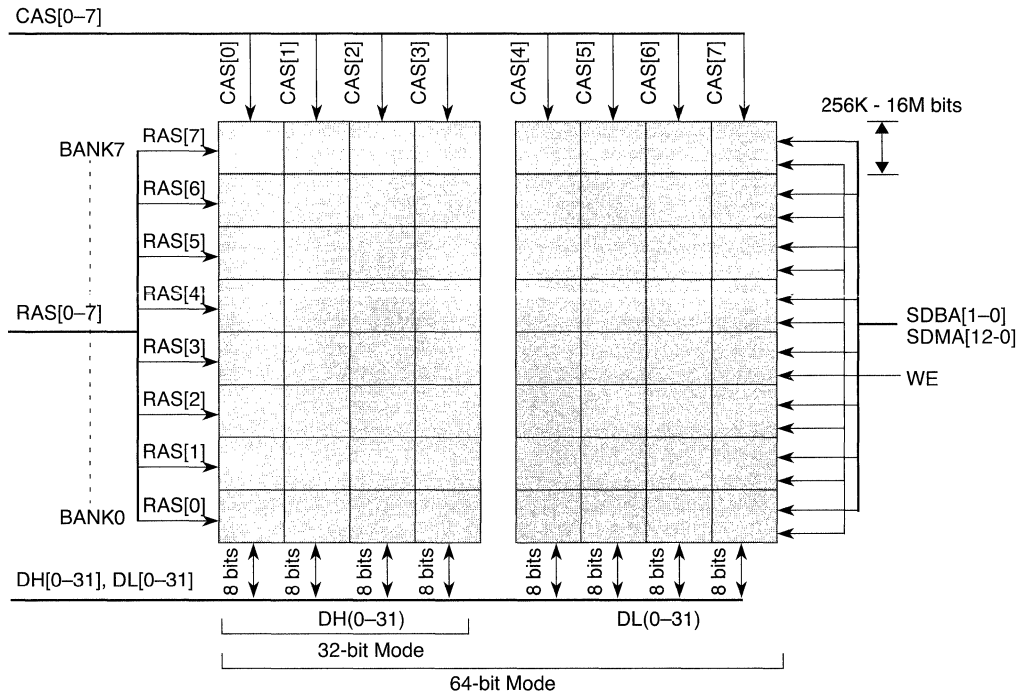


Figure 6-4. DRAM Memory Organization

6.3.1 Supported FPM or EDO DRAM Organizations

It is not necessary to use identical memory chips in each memory bank; individual memory banks may be of differing sizes, but not of different type (SDRAM). The MPC8240 can be configured to provide 9–13 row bits to a particular bank, and 7–12 column bits. Table 6-4 summarizes the DRAM configurations supported by the MPC8240. This table is not exhaustive, although it covers most available DRAMs.

Note

The largest DRAM that can be supported is limited to 24 total address bits.

The MPC8240 can be configured at system start-up by using a memory-polling algorithm or hardcode in a boot ROM, to correctly map the size of each bank in memory. The MPC8240 uses its bank map to assert the appropriate $\bar{R}AS[0-7]$ signals for memory accesses according to the provided bank depths.

System software must also configure MCCR1 register in the MPC8240 at system start-up to appropriately multiplex the row and column address bits for each bank for the devices being used as shown in Table 6-4.

Note

Any unused banks should have their starting and ending addresses programmed out of the range of memory banks in use. Otherwise memory may become corrupted in the overlapping address range. Unused memory banks' starting and ending addresses should always be mapped to memory space that is not used by the system.

Table 6-3 shows the unsupported multiplexed row and column address bit configurations in the 32- and 64-bit data bus mode. They result in non-contiguous address spaces.

Table 6-3. Unsupported Multiplexed Row and Column Address Bits

32-bit Data Bus Mode	64-bit Data Bus Mode
13x10	13x10
13x9	13x9
13x8	13x8
12x8	–
12x7	12x7
11x8	–
11x7	11x7
10x8	–

Table 6-4. Supported FPM or EDO DRAM Device Configurations

DRAM Devices	Devices (64-bit Bank)	Device Configuration	Row x Column Bits	64-bit Bank Size (Mbytes)	8 Banks of Memory (Mbytes)
4 Mbits	4	256 Kbits x 16	9 x 9	2	16
	4	256 Kbits x 16	10 x 8 (64-bit only)	2	16
	8	512 Kbits x 8	10 x 9	4	32
	8	512 Kbits x 8	11 x 8 (64-bit only)	4	32
	16	1 Mbits x 4	10 x 10	8	64
	16	1 Mbits x 4	11 x 9	8	64
	64	4 Mbits x 1	12 x 10	32	256
	64	4 Mbits x 1	11 x 11	32	256

Table 6-4. Supported FPM or EDO DRAM Device Configurations

DRAM Devices	Devices (64-bit Bank)	Device Configuration	Row x Column Bits	64-bit Bank Size (Mbytes)	8 Banks of Memory (Mbytes)
16 Mbits	2	512 Kbits x 32	11 x 8	4	32
	2	512 Kbits x 32	10 x 9	4	32
	4	1 Mbits x 16	12 x 8 (64-bit only)	8	64
	4	1 Mbits x 16	11 x 9	8	64
	4	1 Mbits x 16	10 x 10	8	64
	8	2 Mbits x 8	12 x 9	16	128
	8	2 Mbits x 8	11 x 10	16	128
	16	4 Mbits x 4	12 x 10	32	256
	16	4 Mbits x 4	11 x 11	32	256
	64	16 Mbits x 1	13 x 11	128	1024
	64	16 Mbits x 1	12 x 12	128	1024
64 Mbits	2	2 Mbits x 32	12 x 9	16	128
	2	2 Mbits x 32	11 x 10	16	128
	4	4 Mbits x 16	12 x 10	32	256
	4	4 Mbits x 16	11 x 11	32	256
	8	8 Mbits x 8	12 x 11	64	512
	16	16 Mbits x 4	13 x 11	128	1024
	16	16 Mbits x 4	12 x 12	128	1024

6.3.2 FPM or EDO DRAM Address Multiplexing

System software must configure the MPC8240 at reset to appropriately multiplex the row and column address bits for each bank. This is done by writing the row address configuration into a memory control configuration register 1 (MCCR1) @ <0xF0> see Table 5-37.

The internal physical addresses $A[0_{msb}-31_{lsb}]$ is multiplexed through the output address pins $SDMA[12-0]$. The row and column bit configuration settings are shown in Figure 6-5 for 32-bit bus mode and Figure 6-6 for 64-bit bus mode. During \overline{RAS} and \overline{CAS} phases, the unshaded row and column bits $SDMA[12-0]$ multiplexes the appropriate physical addresses.

6.3.2.1 Row Bit Multiplexing During (\overline{RAS}) The Row Phase

The following list shows the relationships between the internal physical addresses $A[0_{msb}-31_{lsb}]$ and the external address pins $SDMA[12-0]$.

FPM or EDO DRAM Interface Operation

- In the 32-bit data bus mode, SDMA12 contains A[6]
- In the 64-bit data bus mode SDMA12 contains A[5]
- If the FPM or EDO has 9 row bits, SDMA[8–0] contains A[12–20].
- If the FPM or EDO has 10 row bits, SDMA[9–0] contains A[11–20].
- If the FPM or EDO has 11 row bits, SDMA[10–0] contains A[10–20].
- If the FPM or EDO has 12 or 13 row bits, SDMA[11–0] contains A[9–20].

Note

SDMA12 is only used as the most-significant row address bit for 13 x 11 FPM or EDO arrays.

6.3.2.2 Column Bit Multiplexing During ($\overline{\text{CAS}}$) the Column Phase

- In the in 32-bit bus mode, SDMA[7–0] contains A[22–29]
- In the 64-bit bus mode, SDMA[7–0] contains A[21–28]

Note

The encoding of SDMA[11–8] depends on the bus mode selected (32- or 64-bit) and on the number of row bits set in MCCR1. See Table 6-5.

Table 6-5. SDMA[11–8] Encodings for 32- and 64-Bit Bus Modes

Row Bits	32-Bit Bus Mode	64-Bit Bus Mode
9	A[9–11, 21]	A[8–11]
10	A[8–10, 21]	A[7–10]
11	A[7–9, 21]	A[6–9]
12	A[6–8, 21]	A[5–8]
13	A[unused, 7–8, 21]	A[unused, 6–8]

6.3.2.3 Graphical View of the Row and Column Bit Multiplexing

See Figure 6-5 and Figure 6-6 for a graphical view of the row and column bit multiplexing.

Row x Col		Physical Address																													
		msb																													lsb
		0-5	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			
13x11	RAS					12			11	10	9	8	7	6	5	4	3	2	1	0											
	CAS						10	9											8	7	6	5	4	3	2	1	0				
12x12	RAS							11	10	9	8	7	6	5	4	3	2	1	0												
	CAS						11	10	9										8	7	6	5	4	3	2	1	0				
12x11	RAS							11	10	9	8	7	6	5	4	3	2	1	0												
	CAS							10	9										8	7	6	5	4	3	2	1	0				
12x10	RAS							11	10	9	8	7	6	5	4	3	2	1	0												
	CAS							9											8	7	6	5	4	3	2	1	0				
12x9	RAS							11	10	9	8	7	6	5	4	3	2	1	0												
	CAS																		8	7	6	5	4	3	2	1	0				
11x11	RAS								10	9	8	7	6	5	4	3	2	1	0												
	CAS								10	9									8	7	6	5	4	3	2	1	0				
11x10	RAS								10	9	8	7	6	5	4	3	2	1	0												
	CAS								9										8	7	6	5	4	3	2	1	0				
11x9	RAS								10	9	8	7	6	5	4	3	2	1	0												
	CAS																		8	7	6	5	4	3	2	1	0				
10x10	RAS									9	8	7	6	5	4	3	2	1	0												
	CAS									9									8	7	6	5	4	3	2	1	0				
10x9	RAS									9	8	7	6	5	4	3	2	1	0												
	CAS																		8	7	6	5	4	3	2	1	0				
9x9	RAS										8	7	6	5	4	3	2	1	0												
	CAS																		8	7	6	5	4	3	2	1	0				

Figure 6-5. DRAM Address Multiplexing SDMA[12-0]—32 Bit Mode

Row x Col		Physical Address																													
		msb																													lsb
		0-4	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		
13x11	RAS				12				11	10	9	8	7	6	5	4	3	2	1	0											
	CAS					10	9	8													7	6	5	4	3	2	1	0			
12x12	RAS							11	10	9	8	7	6	5	4	3	2	1	0												
	CAS				11	10	9	8													7	6	5	4	3	2	1	0			
12x11	RAS							11	10	9	8	7	6	5	4	3	2	1	0												
	CAS				10	9	8														7	6	5	4	3	2	1	0			
12x10	RAS							11	10	9	8	7	6	5	4	3	2	1	0												
	CAS					9	8														7	6	5	4	3	2	1	0			
12x9	RAS							11	10	9	8	7	6	5	4	3	2	1	0												
	CAS						8														7	6	5	4	3	2	1	0			
12x8	RAS							11	10	9	8	7	6	5	4	3	2	1	0												
	CAS																				7	6	5	4	3	2	1	0			
11x11	RAS								10	9	8	7	6	5	4	3	2	1	0												
	CAS					10	9	8													7	6	5	4	3	2	1	0			
11x10	RAS								10	9	8	7	6	5	4	3	2	1	0												
	CAS						9	8													7	6	5	4	3	2	1	0			
11x9	RAS								10	9	8	7	6	5	4	3	2	1	0												
	CAS							8													7	6	5	4	3	2	1	0			
11x8	RAS								10	9	8	7	6	5	4	3	2	1	0												
	CAS																				7	6	5	4	3	2	1	0			
10x10	RAS									9	8	7	6	5	4	3	2	1	0												
	CAS								9	8											7	6	5	4	3	2	1	0			
10x9	RAS									9	8	7	6	5	4	3	2	1	0												
	CAS								8												7	6	5	4	3	2	1	0			
10x8	RAS									9	8	7	6	5	4	3	2	1	0												
	CAS																				7	6	5	4	3	2	1	0			
9x9	RAS										8	7	6	5	4	3	2	1	0												
	CAS									8											7	6	5	4	3	2	1	0			

Figure 6-6. DRAM Address Multiplexing SDMA[12-0]—64 Bit Mode

6.3.3 FPM or EDO Memory Data Interface

The MPC8240 supports flow-through data-path buffering for the DRAM data interface between the internal processor bus and external memory data bus, which must be

configured as described in Table 6-6 and Table 6-7. Unspecified bit settings have undefined behavior.

Table 6-6. FPM or EDO Memory Parameters

Bit Name	Register and Offset	Bit Number in Register
RAM_TYPE	MCCR1 @ 0xF0	17
EDO	MCCR2 @ 0xF4	16
PCKEN	MCCR1 @ 0xF0	16
WRITE_PARITY_CHK_EN	MCCR2 @ 0xF4	19
INLINE	MCCR4 @ 0xFC	22
REGISTERED	MCCR4 @ 0xFC	20
RMW_PAR	MCCR2 @ 0xF4	0
ECC_EN	MCCR2 @ 0xF4	17
MEM_PERR_EN	ErrEnR1 @ 0xF0	2
MB_ECC_ERR_EN	ErrEnR2 @ 0xF4	3

Table 6-7. FPM or EDO System Configurations

RAM TYPE	EDO	PCKEN	WRITE PARITY CHKEN	INLINE	REGISTERED	RMW PAR	ECC EN	MEM PERR EN	Description
1	0	0	0	0	0	0	0	0	FPM, flow-through, no ECC or parity
1	1	0	0	0	0	0	0	0	EDO, flow-through, no ECC or parity
1	0	1	0	0	0	0	0	1	FPM, flow-through, parity enabled
1	1	1	0	0	0	0	0	1	EDO, flow-through, parity enabled
1	0	0	0	0	0	0	1	1	FPM, flow-through, ECC enabled
1	1	0	0	0	0	0	1	1	EDO, flow-through, ECC enabled
1	0	1	0	0	0	1	0	1	FPM, RMW_PAR
1	1	1	0	0	0	1	0	1	EDO, RWM_PAR

The data-path between the internal processor bus to the external memory bus in flow-through mode is shown in Figure 6-7. The flow-through mode is the default data bus buffering mode for the MPC8240.

Note

In-line ECC is not available with FPM or EDO DRAM.

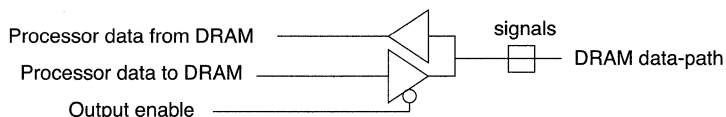


Figure 6-7. FPM-EDO Flow-through Memory Interface

6.3.4 FPM or EDO DRAM Initialization

At system reset, the main memory is inactive. When the system reset signals (HRST_CPU and HRST_CTRL) are negated, the MEMGO bit is cleared to zero (0) which turns off the memory controller. The processor core starts fetching boot code from ROM (local or PCI). For systems containing FPM or EDO DRAM, the boot code must set the MPC8240 configuration bit RAMTYP = 1.

Additionally, all other MPC8240 configuration registers relevant to DRAM must be initialized. Table 6-8 shows the register fields in the memory interface configuration registers (MICR) and the memory control configuration registers (MCCR) @ <hh>.

Table 6-8. Memory Interface Configuration Register Fields

Register Field	Description	Configuration Register
RAMTYP	SDRAM, FPM, or EDO DRAM	MCCR1 @ <F0>
Memory Bank Start and End Addresses		MICR @ <80>-<9C>
Memory Bank Enables		MICR @ <A0>
Row Address Bits For Each Bank		MCCR1 @ <F0>
PCKEN	Parity check enable	MCCR1 @ <F0>
SREN	Self refresh enable	MCCR1 @ <F0>
REFINT	Interval between refreshes	MCCR2 @ <F4>
RP1	RAS precharge interval	MCCR3 @ <F8>
RCD2	RAS to CAS delay	MCCR3 @ <F8>
CAS3	CAS assertion interval for first data beat	MCCR3 @ <F8>
CP4	CAS precharge interval	MCCR3 @ <F8>
CAS5	CAS assertion interval for page mode data beats	MCCR3 @ <F8>
RAS6P	RAS assertion interval for CBR refresh	MCCR3 @ <F8>
RMW_PAR	Read modify write parity	MCCR2 @ <F4>
ECC_EN	ECC enable	MCCR2 @ <F4>
REGISTERED	Register data-path = 0 (off)	MCCR4 @ <FC>

Table 6-8. Memory Interface Configuration Register Fields (Continued)

Register Field	Description	Configuration Register
INLINE	In-line data-path = 0 (off)	MCCR4 @ <FC>
WRITE_PARITY_CHK_EN	Enable write path parity error reporting	MCCR2 @ <F4>

After configuration of all these parameters is complete, the system software must set the configuration bit MEMGO which turns on the memory controller. The MPC8240 will perform one $\overline{\text{CAS}}$ before $\overline{\text{RAS}}$ (CBR) refresh cycle each time REFINT elapses. After eight refreshes, the main memory array is available for read and write accesses.

6.3.5 FPM or EDO DRAM Interface Timing

The read and write timing for DRAM is also controlled through programmable registers. These registers are programmed by system software at system start up to control:

- $\overline{\text{RAS}}$ precharge time (RP_1)
- $\overline{\text{RAS}}$ to $\overline{\text{CAS}}$ delay time (RCD_2)
- $\overline{\text{CAS}}$ pulse width for the first access (CAS_3)
- $\overline{\text{CAS}}$ precharge time (CP_4)
- $\overline{\text{CAS}}$ pulse width in page mode (CAS_5)

All signal transitions occur on system clock rising edges. Figure 6-9 shows DRAM read timing with the programmable variables. Figure 6-11 shows DRAM write timing with the programmable variables. As shown, the provided timing variables are applicable to both read and write timing configuration. System software is responsible for optimal configuration of these parameters after reset. This configuration process must be completed at system start up before any attempts to access DRAM. The actual values used by boot code depend on the memory technology used.

Note

No more than 8 PCI clock cycles should elapse between successive assertions of $\overline{\text{CAS}}$ within a burst.

Table 6-9 defines the timing parameters for FPM or EDO DRAM. Subscripts identify timing variables.

Table 6-9. FPM or EDO Timing Parameters

Symbol	Timing Parameter
AA	Access time from column address
ASC	Column address setup time
ASR	Row address setup time
CAC	Access time from $\overline{\text{CAS}}$
CAH	Column address hold time
CAS ₃	$\overline{\text{CAS}}$ assertion interval for a single beat, or for the first access in a burst
CAS ₅	$\overline{\text{CAS}}$ assertion interval for page mode access
CHR	$\overline{\text{CAS}}$ hold time for CBR refresh
CP	$\overline{\text{CAS}}$ precharge time
CP ₄	$\overline{\text{CAS}}$ precharge interval during page-mode access
CRP	$\overline{\text{CAS}}$ to $\overline{\text{RAS}}$ precharge time
CSH	$\overline{\text{CAS}}$ hold time
CSP	$\overline{\text{CAS}}$ setup time for CBR refresh
DH	Data in hold time
DS	Data in setup time
PC	Fast page mode cycle time
RAC	Access time from $\overline{\text{RAS}}$
RAD	$\overline{\text{RAS}}$ to column address delay time
RAH	Row address hold time
RAL	Column address to $\overline{\text{RAS}}$ lead time
RAS _{6P}	$\overline{\text{RAS}}$ assertion interval for CBR refresh
RASP	$\overline{\text{RAS}}$ pulse width (page-mode)
RASS	Self-refresh interval (power saving modes only)
RC	Random access (read, write, or refresh) cycle time
RCD ₂	$\overline{\text{RAS}}$ to $\overline{\text{CAS}}$ delay interval
RHCP	$\overline{\text{RAS}}$ hold time from $\overline{\text{CAS}}$ precharge (page mode only)
RP ₁	$\overline{\text{RAS}}$ precharge interval
RPC	$\overline{\text{RAS}}$ precharge to $\overline{\text{CAS}}$ active time
RSH	$\overline{\text{RAS}}$ hold time
WCH	Write command hold time (referenced to $\overline{\text{CAS}}$)
WCS	Write command setup time
WP	Write command pulse width
WRH	Write to $\overline{\text{RAS}}$ hold time (CBR refresh)
WRP	Write to $\overline{\text{RAS}}$ precharge time (CBR refresh)

Note

All signal transitions occur on the rising edge of the memory bus clock.

Figure 6-8 through Figure 6-13 shows FPM or EDO timing for various types of accesses with ECC disabled.

Figure 6-8 shows a single-beat read operation.

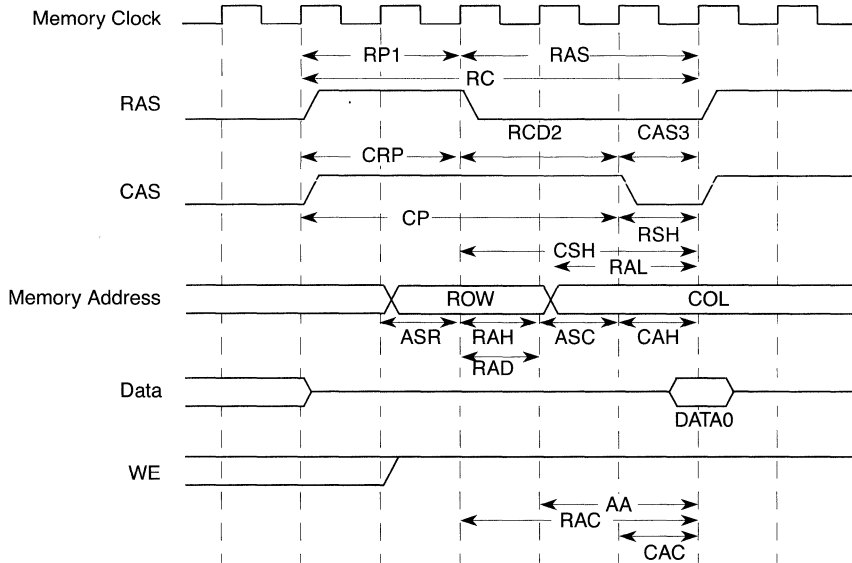


Figure 6-8. DRAM Single-Beat Read Timing (No ECC)

Figure 6-9 shows a 64-bit bus mode burst read operation.

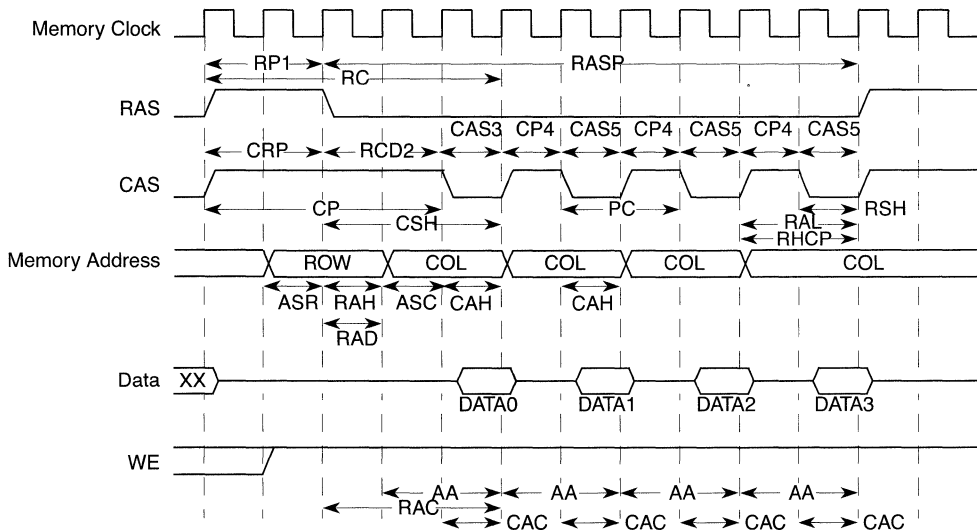


Figure 6-9. DRAM Four-Beat Burst Read Timing (No ECC)—64-Bit Mode

FPM or EDO DRAM Interface Operation

Figure 6-10 shows a 32-bit bus mode burst read operation.

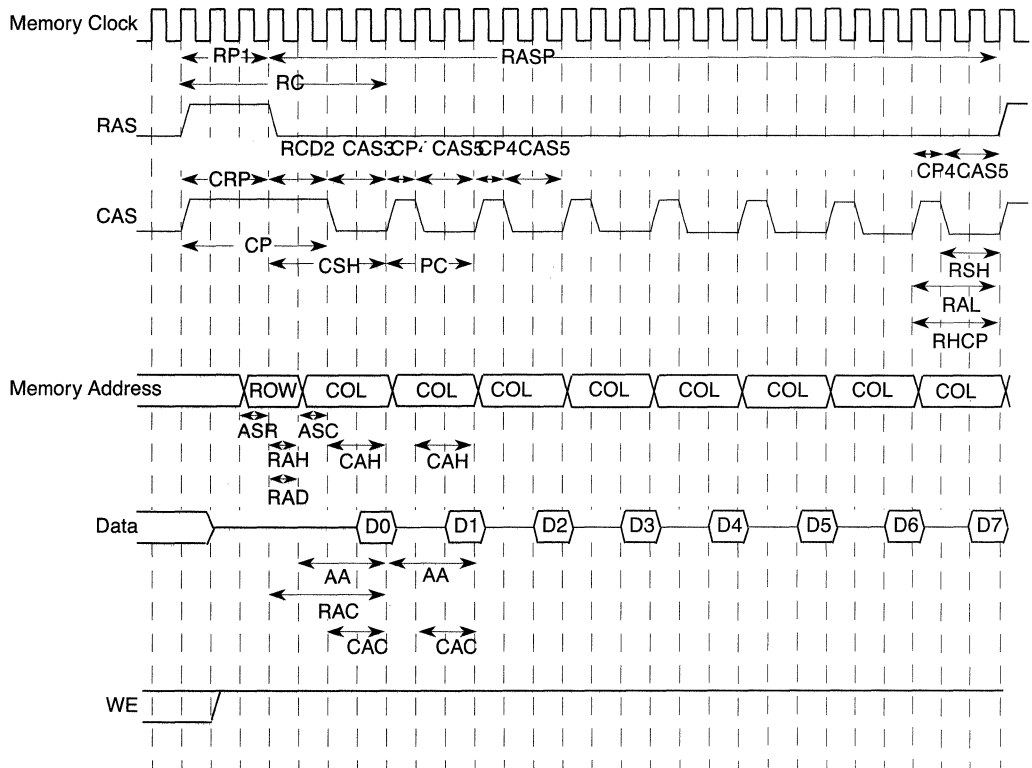


Figure 6-10. DRAM Eight-Beat Burst Read Timing Configuration—32-Bit Mode

Figure 6-11 shows a single-beat write operation.

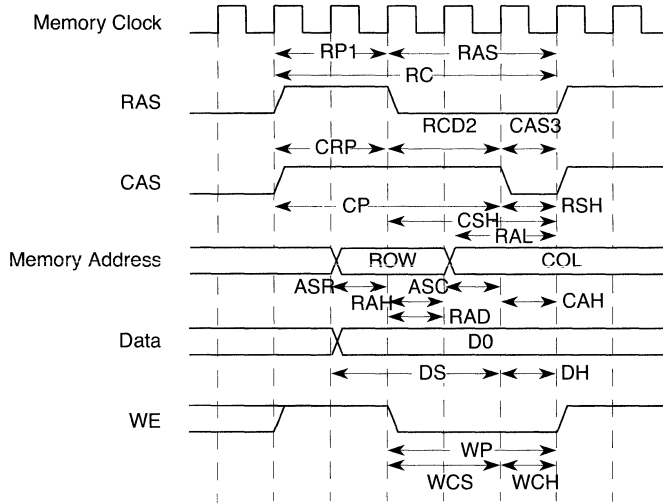


Figure 6-11. DRAM Single-Beat Write Timing (No ECC)

Figure 6-12 shows a 64-bit bus mode burst write operation.

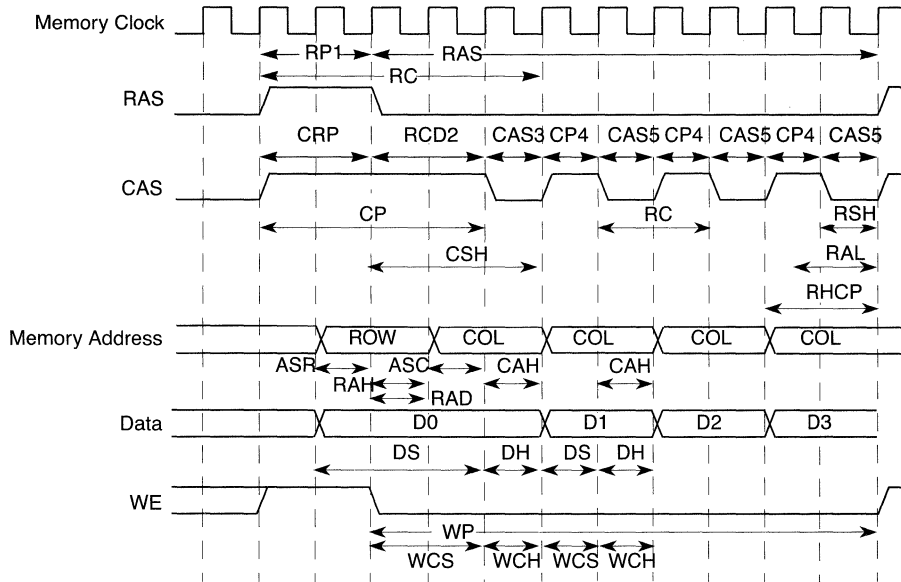


Figure 6-12. DRAM Four-Beat Burst Write Timing (No ECC)—64-Bit Mode

Figure 6-13 shows a 32-bit bus mode burst write operation.

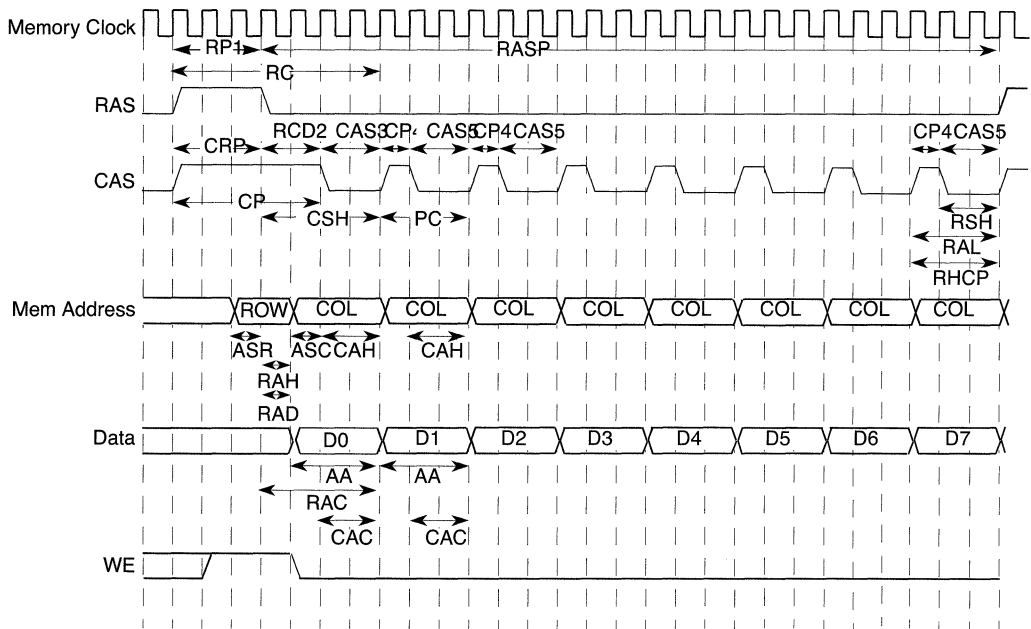


Figure 6-13. DRAM Eight-beat Burst Write Timing (No ECC)—32 Bit Mode

6.3.6 DMA Burst Wrap

The MPC8240 supports burst-of-four data beats for accesses with a 64-bit data path and burst-of-eight data beats for accesses with a 32-bit data path. The burst is always sequential, and the critical double word is always supplied first as detailed in the following two examples:

Note

A double word (DW) is 64 bits and a word (W) is 32 bits. DW2 is the third double word in the 64-bit example and W4–W5 is the third double word in the 32-bit example.

- Using a 64-bit data path, if the local processor requests the third double word (DW2) of a cache line, the MPC8240 reads double words from memory in the order DW2-DW3-DW0-DW1.
- Using a 32-bit data path, if the local processor requests the third double word (W4 and W5) of a cache line, the MPC8240 reads words from memory in the order W4-W5-W6-W7-W0-W1-W2-W3.

6.3.7 FPM or EDO DRAM Page Mode Retention

Under certain conditions, the MPC8240 retains the currently active FPM or EDO page by holding $\overline{\text{RAS}}$ asserted for pipelined burst accesses. These conditions are as follows:

- A pending transaction (read or write) hit the active FPM or EDO page.
- There are no pending refreshes.
- The maximum $\overline{\text{RAS}}$ assertion interval (controlled by PGMAX) has not been exceeded.

Page mode can dramatically reduce access latencies for page hits. Depending on the memory system design and timing parameters, using page mode can save three to four clock cycles from subsequent burst accesses that hit in an active page. Page mode is disabled by clearing the PGMAX parameter (PGMAX = 0x00) located in the memory page mode register <0xA3>.

6.3.8 FPM or EDO DRAM Parity and RMW Parity

When configured for FPM or EDO, the MPC8240 supports two forms of parity checking and generation—normal parity and read-modify-write (RMW) parity. Normal parity assumes that each of the eight parity bits is controlled by a separate $\overline{\text{CAS}}$ signal. Thus, for a single-beat write from PCI to system memory, the MPC8240 generates a parity bit for each byte written to memory.

RMW parity assumes that all eight parity bits are controlled by a single $\overline{\text{CAS}}$ signal; therefore it must be written as a single 8-bit quantity (byte). Therefore, for any write operation to system memory that is less than a double word, the MPC8240 must latch the write data, read an entire 64-bit double word from memory, check the parity of that double word, merge the write data with that double word, regenerate parity for the new double word, and finally write the new double word back to memory.

The MPC8240 checks parity on all memory reads, provided parity checking is enabled (PCKEN = 1). The MPC8240 generates parity for the following operations:

- PCI-to-memory write operations
- Local processor single-beat write operations with RMW parity enabled (RMW_PAR = 1)

The local processor is expected to generate parity for all other memory write operations as the data goes directly to memory and does not pass through the MPC8240.

Note

The MPC8240 does not support RMW parity mode when in 32-bit data-path mode (RMW_PAR = 0).

6.3.8.1 RMW Parity Latency Considerations

When RMW parity is enabled, the time required to read, modify, and write increases latency for some transactions.

For local processor single-beat writes to system memory, the MPC8240 latches the data, performs a double-word read from system memory (checking parity), and then merges the write data from the processor with the data read from memory. The MPC8240 then generates new parity bits for the merged double word and writes the data and parity to memory. The read-modify-write process adds six clock cycles to a single-beat write operation. If page-mode retention is enabled ($PGMAX > 0$), then the MPC8240 keeps the memory in page-mode for the read-modify-write sequence. Figure 6-16 shows FPM or EDO timing for a local processor single-beat write operation with RMW parity enabled.

For PCI writes to system memory with RMW parity enabled, the MPC8240 latches the data in the internal PCI-to-system-memory-write buffer (PCMWB). If the PCI master writes complete double words to system memory, the MPC8240 generates the parity bits when the PCMWB is flushed to memory. However, if the PCI master writes 32-, 16-, or 8-bit data that cannot be gathered into a complete double word in the PCMWB, a read-modify-write operation is required. The MPC8240 performs a double-word read from system memory (checking parity), and then merges the write data from the PCI master with the data read from memory. The MPC8240 then generates new parity for the merged double word and writes the data and parity to memory. If page mode retention is enabled ($PGMAX > 0$), the MPC8240 keeps the memory in page mode for the read-modify-write sequence.

Because the local processor drives all eight parity bits during burst writes to system memory, these transactions go directly to the DRAMs with no performance penalty. All other transactions are unaffected and operate as in normal parity mode.

6.3.9 FPM or EDO ECC

As an alternative to simple parity, the MPC8240 supports ECC for the data-path between the MPC8240 and system memory. ECC not only allows the MPC8240 to detect errors in the memory data-path but also to correct single-bit errors in the 64-bit data-path. The ECC logic in the MPC8240 detects and corrects all single-bit errors and detects all double-bit errors and all errors within a nibble. Other errors may be detected but are not guaranteed to be either detected or corrected. Multiple-bit errors are always reported when detected. However, when a single-bit error occurs, the single-bit error counter register is incremented, and its value is compared to the single-bit error trigger register. If the values are not equal, no error is reported; if the values are equal, then an error is reported. Thus, the single-bit error registers may be programmed so that minor faults with memory are corrected and ignored, but a catastrophic memory failure generates an interrupt. The syndrome equations for the ECC code are shown in Table 6-10. For supported configurations, see Table 6-4.

The MPC8240 supports concurrent ECC for the FPM or EDO data-path and parity for the

6.3.9.1 FPM or EDO DRAM Interface Timing with ECC

When ECC is enabled, the time required to check and generate the ECC codes increases access latency. Figure 6-14 through Figure 6-16 shows FPM or EDO timings for various types of accesses with ECC enabled.

For processor burst reads from system memory, checking the ECC codes for errors requires an additional two clock cycles for the first data returned and at least four clock cycles for subsequent beats. These requirements do not depend on the buffer type or whether FPM or EDO is used for system memory.

For local-processor single-beat writes to system memory, the MPC8240 latches the data, performs a double-word read from system memory (checking and correcting any ECC errors), and merges the write data from the processor with the data read from memory. The MPC8240 then generates a new ECC code for the merged double word and writes the data and ECC code to memory. This read-modify-write process adds six clock cycles to a single-beat write operation. If page-mode retention is enabled ($PGMAX > 0$), the MPC8240 keeps the memory in page mode for the read-modify-write sequence.

For local-processor burst writes to system memory, the MPC8240 latches the data in the internal copy-back buffer and flushes the buffer to memory at the earliest opportunity. The MPC8240 generates the ECC codes when the flush occurs.

Note

The MPC8240 does not check the data being overwritten in memory.

For PCI writes to system memory with ECC enabled, the MPC8240 latches the data in the internal PCI to memory write buffer (PCMWB). If the PCI master writes complete double words to system memory, the MPC8240 generates the ECC codes when the PCMWB is flushed to memory.

Note

If the PCI master writes 32-, 16-, or 8-bit data that cannot be gathered into a complete double word in the PCMWB, a read-modify-write operation is required.

The MPC8240 performs a double word read from system memory (checking and correcting any ECC errors), then merges the write data from the PCI master with the data read from memory, generates a new ECC code for the merged double word,

and writes the data and ECC code to memory.

Figure 6-14 shows a FPM burst read operation.

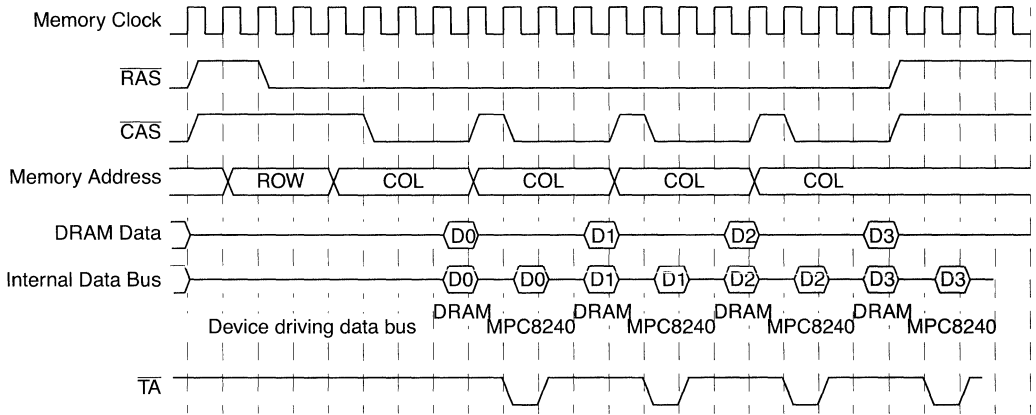


Figure 6-14. FPM DRAM Burst Read with ECC

Figure 6-15 shows an EDO burst read operation

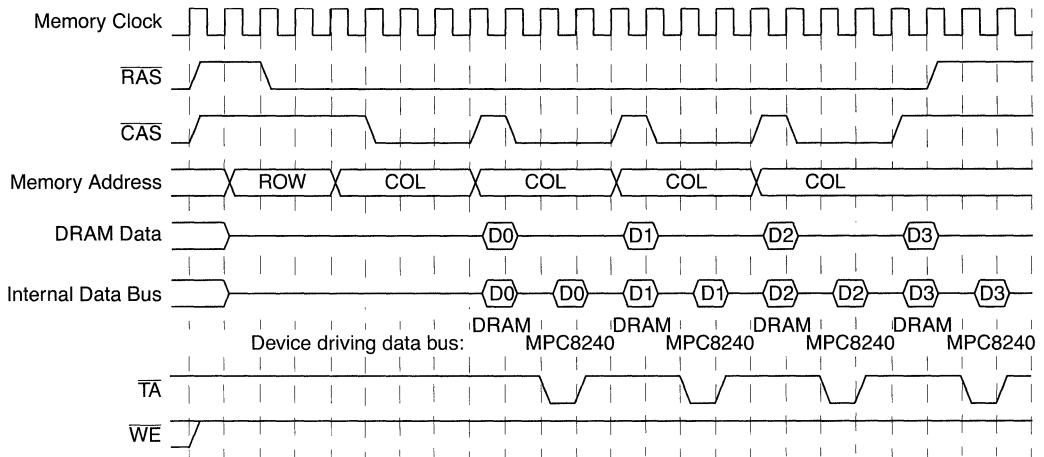


Figure 6-15. EDO DRAM Burst Read Timing with ECC

Figure 6-16 shows a single-beat write operation.

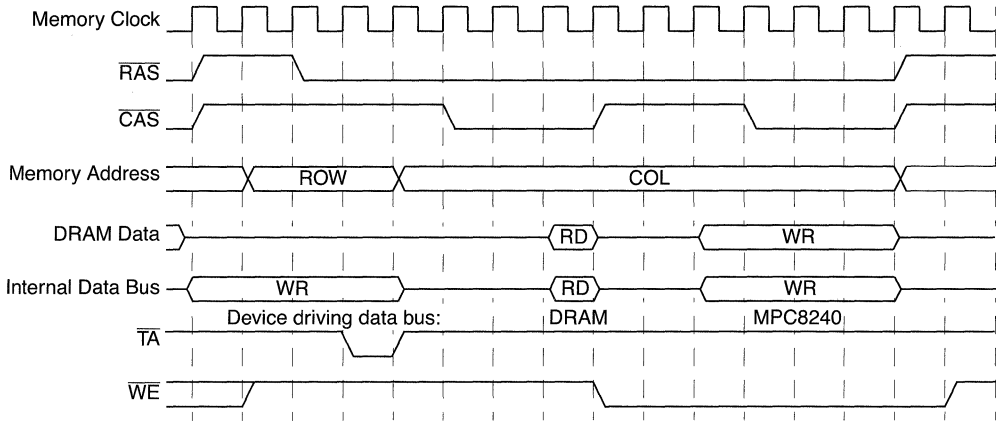


Figure 6-16. DRAM Single-Beat Write Timing with RMW or ECC Enabled

6.3.10 FPM or EDO DRAM Refresh

The MPC8240's memory interface distributes $\overline{\text{CAS}}$ -before- $\overline{\text{RAS}}$ (CBR) refreshes to DRAM according to the interval specified in its REFINT configuration register. This configuration register must be programmed by boot code at system start up. The value to be stored in REFINT represents the number of memory clock cycles required between CBR refreshes.

This value should allow for a potential collision between memory access and refresh. (The per row refresh interval should be reduced by the longest memory access time.) For example, for a DRAM with a cell refresh time of 64 mS and 4096 rows, the per row refresh interval would be $64 \text{ mS} / 4,096 \text{ rows} = 15.6 \mu\text{S}$. If the memory interface runs at 66 MHz, $15.6 \mu\text{S}$ represents 1,030 memory clock cycles. If a burst read is in progress when a refresh is to be performed, the refresh waits for the read to complete. Thus, the per-row refresh interval (1,030 clocks) should be reduced by the longest access time (based on configuration parameters) and then stored to REFINT (as a binary representation of the difference).

The duration of $\overline{\text{RAS}}$ assertion during a CBR refresh is controlled by configuration parameter RAS_{6P} . Refer to Figure 6-17 and Table 6-12 for further information.

6.3.10.1 FPM or EDO Refresh Timing

The refresh timing for DRAM is controlled through $\text{MCCR3}[\text{RAS}_{6P}]$. This register is initialized during reset and controls the $\overline{\text{RAS}}$ active time during a CBR refresh. (Refer to interval RAS_{6P} in Figure 6-17.) As shown, the MPC8240 implements bank staggering for CBR refreshes. System software is responsible for optimal configuration of interval RAS_{6P} after system start-up. Such configuration must be completed before attempting access to DRAM.

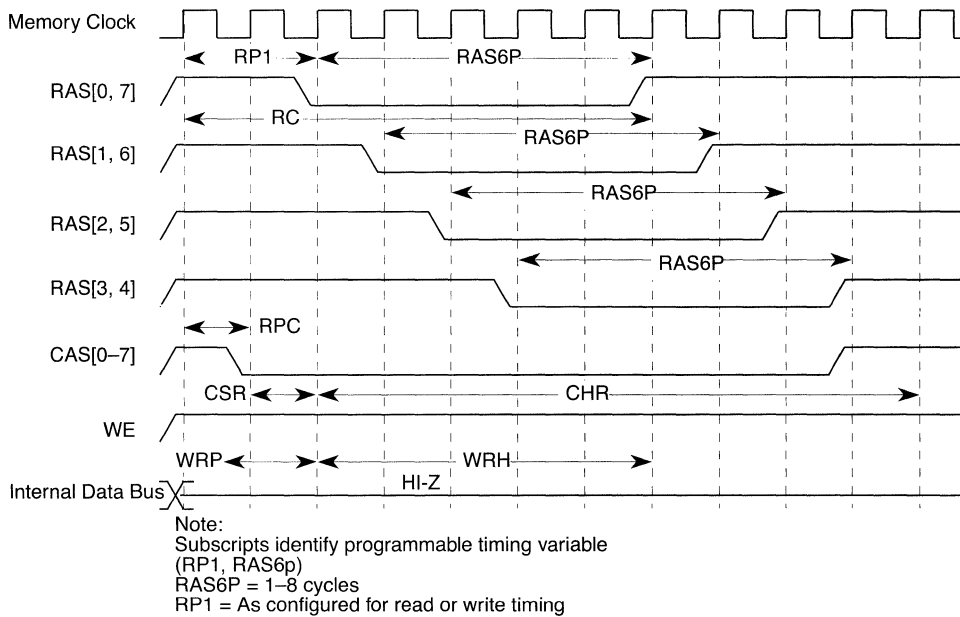


Figure 6-17. DRAM Bank Staggered CBR Refresh Timing Configuration

6.3.11 FPM or EDO DRAM Power Saving Modes

The MPC8240’s memory interface provides for sleep, doze, and nap power saving modes defined for the processor core. In sleep mode, the MPC8240 can be configured to take advantage of DRAM self-refresh mode, to provide normal refresh to DRAM, or to provide no refresh support. If the MPC8240 is configured to provide no refresh support in sleep mode, system software must appropriately preserve DRAM data, that is by copying to disk. In doze and nap modes, the MPC8240 supplies normal CBR refresh to DRAM. Table 6-12 provides a summary of the MPC8240 configuration bits relevant to power saving modes. In Table 6-12, PMCR refers to the MPC8240’s power management configuration register, and MEMCFG refers to memory configuration register.

Table 6-12. FPM or EDO DRAM Power Saving Modes Refresh Configuration

Power Saving Mode	Refresh Type	Power Management Control Register (PMCR)					MCCR1 SREN
		PM	DOZE	NAP	SLEEP	LP_REF_EN	
Doze	Normal	1	1	0	0	—	—
Nap	Normal	1	—	1	0	—	—
Sleep	Self	1	—	—	1	1	1
	Normal	1	—	—	1	1	0
	None	1	—	—	1	0	—

6.3.11.1 Self-Refresh in Sleep Mode

The MPC8240 allows the system designer to use DRAMs that provide self refresh for power-down situations. These DRAMs should be used if data retention is imperative during sleep mode. The MPC8240 properly configures these DRAMs during sleep mode if the appropriate configuration register bit is set. The timing for such a self refresh initiation is shown in Figure 6-18.

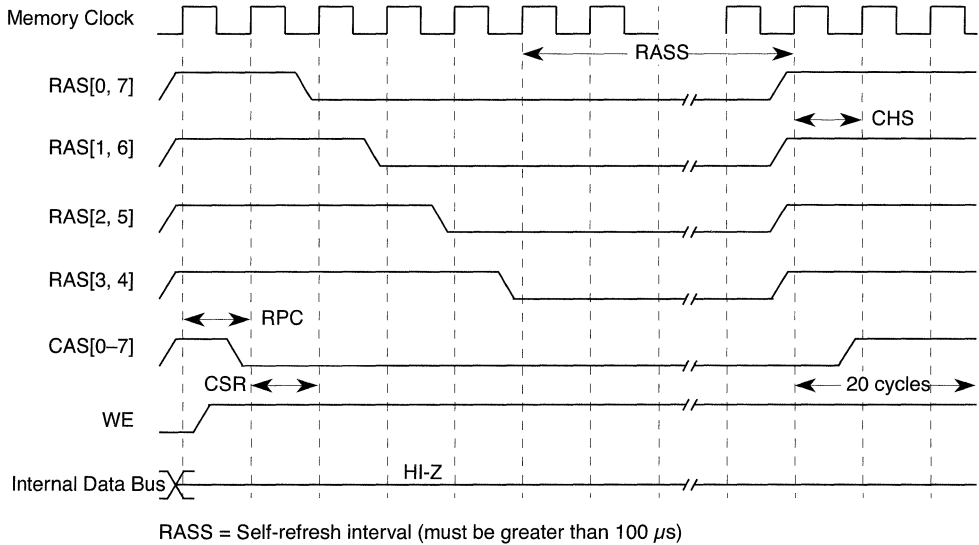


Figure 6-18. DRAM Self-Refresh Timing Configuration

6.4 SDRAM Interface Operation

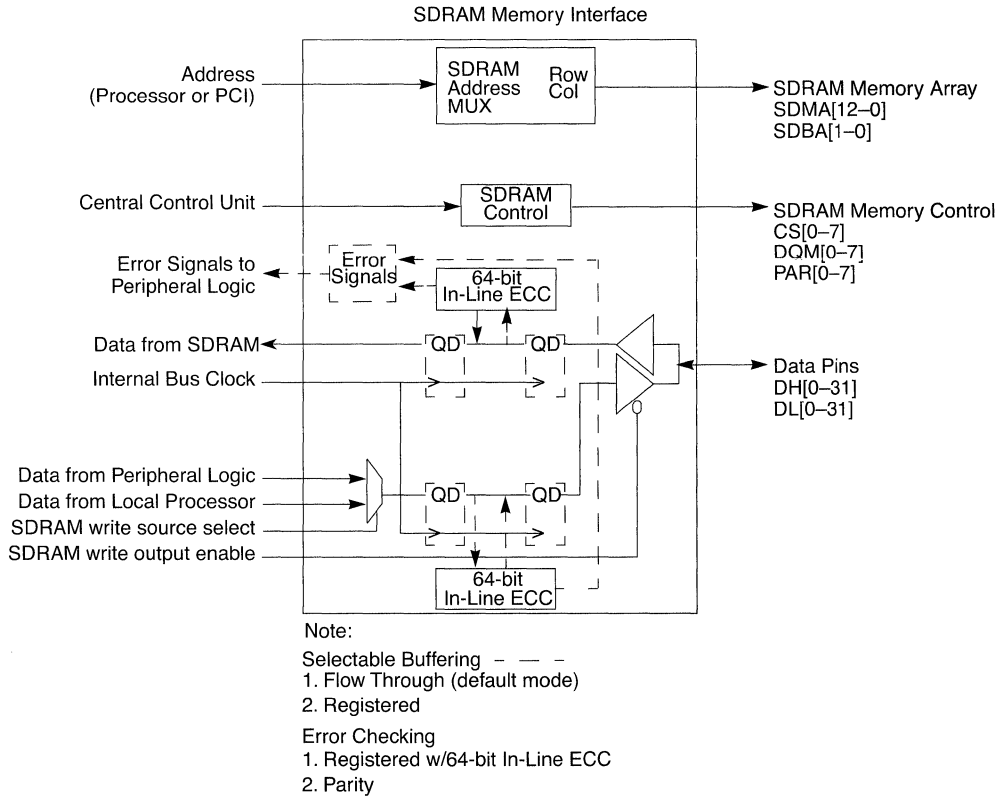


Figure 6-19. SDRAM Memory Interface Block Diagram

The MPC8240 provides control functions and signals for JEDEC-compliant SDRAM. The MPC8240 supplies the SDRAM_CLK[0-3] to be distributed to the SDRAM. These clocks are the same frequency and in phase with the memory bus clock.

The SDRAM memory bus can be configured to be 64-bits (72 bits with parity) requiring a four-beat SDRAM data burst, or configured to be 32-bits (36 bits with parity) requiring an eight-beat SDRAM data burst.

Thirteen row/column multiplexed address signals (SDMA[12-0]) and two bank select signals (SDBA[1-0]) provide for SDRAM densities up to 128 Mbits. Eight chip select signals ($\overline{CS}[0-7]$) support up to eight banks of memory. Eight SDRAM data in/out mask signals (DQM[0-7]) provide byte selection for 32- and 64-bit accesses.

Note

An 8-bit SDRAM device has a DQM signal and eight data signals (DQ[0–7]). A 16-bit SDRAM device has two DQM signals associated to specific halves of the sixteen data signals (DQ[0–7] and DQ[8–15]).

Figure 6-20 shows the MPC8240's relationships between Data Byte Lane[0–7], DQM[0–7], and DH[0–31] and DL[0–31] for 32- and 64-bit modes.

Data Byte Lane	Data In/Out Mask	Data Bus 32-bit Mode	Data Bus 64-bit Mode
0 _(MSB)	DQM[0]	DH[0–7]	DH[0–7]
1	DQM[1]	DH[8–15]	DH[8–15]
2	DQM[2]	DH[16–23]	DH[16–23]
3	DQM[3]	DH[24–31]	DH[24–31]
4	DQM[4]		DL[0–7]
5	DQM[5]		DL[8–15]
6	DQM[6]		DL[16–23]
7 _(LSB)	DQM[7]		DL[24–31]

Figure 6-20. SDRAM Data Bus Lane Assignments

In addition, there are sixty four data signals (DH[0–31] and DL[0–31]), a write enable signal ($\overline{\text{WE}}$), a row address strobe signal ($\overline{\text{SDRAS}}$), a column address strobe signal ($\overline{\text{SDCAS}}$), a memory clock enable signal (CKE), and eight parity signals (PAR[0–7]).

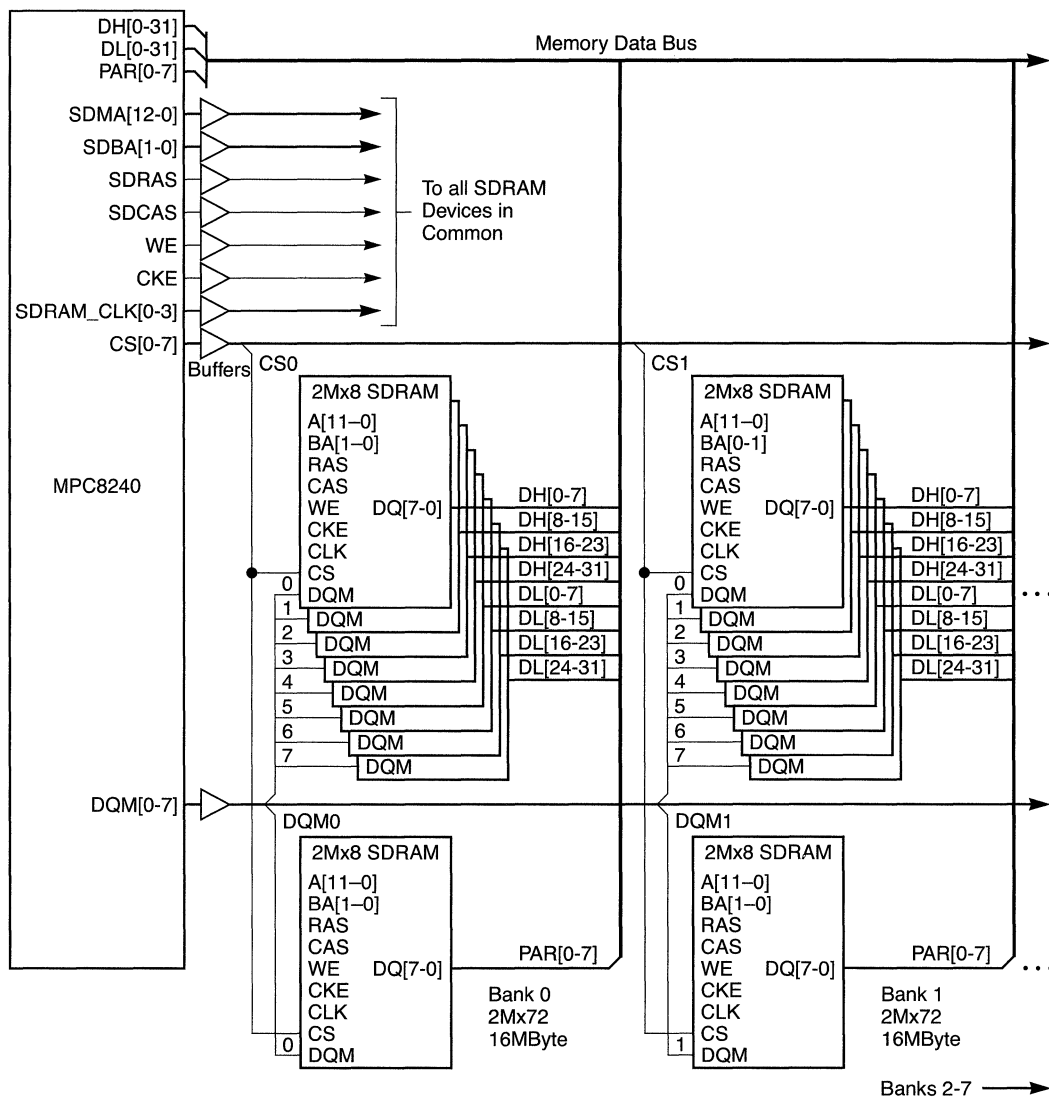
Note

The banks can be built of x1, x4, x8, x16, or x32 SDRAMs as they become available.

Collectively, these interface signals allow a total of 1 Gbyte of addressable memory. Programmable $\overline{\text{CAS}}$ latency is supported for data read operations. For write operations, the first beat of write data is supplied concurrent with the write command. The memory design must be byte-selectable for writes using the MPC8240's DQM outputs.

The MPC8240 allows four simultaneous open pages for page mode; the number of clocks for which the pages are maintained open is programmable by the BSTOPRE parameter. Page register allocation uses a least-recently used (LRU) algorithm.

In addition to 14 address signals, 8 \overline{CS} signals, and 8 DQM signals, there are 64 data signals, a write enable (\overline{WE}) signal, a column address strobe (\overline{SDCAS}) signal, a row address strobe (\overline{SDRAS}), a memory clock enable (CKE) signal, and 8 bidirectional data parity signals. An example SDRAM configuration, with 8 banks, is shown in Figure 6-21. The SDRAM configuration is an eight bank, 128 Mbyte, SDRAM memory array with a 72-bit data bus. Each bank is comprised of nine 2M bit x 8 SDRAMs. One of the nine 2M bit x 8 SDRAMs is used for the bank's parity checking function. Certain address and control lines may or may not require buffering, depending upon the system design. Analysis of the MPC8240 AC specifications, desired memory operating frequency, capacitive loads, and board routing loads can assist the system designer in deciding whether any signals require buffering.



NOTES:

1. All signals are connected in common (in parallel) except for CS[0-7], SDRAM_CLK[0-3], the DQM signals us parity, and the data bus lines.
2. Optional parity memories may use any DQM signal. To minimize loading, a different DQM line is recommend each bank: DQM0 for Bank 0, DQM1 for Bank 1, etc.
3. Each of the CS[0-7] signals correspond with a separate physical bank of memory; CS0 for the first bank, and
4. Buffering may be needed if large memory arrays are used.
5. SDRAM_CLK[0-3] signals may be apportioned among all memory devices.

Figure 6-21. Example 128 Mbyte SDRAM Configuration With Parity

6.4.1 Supported SDRAM Organizations

It is not necessary to use identical memory chips in each memory bank; individual memory banks may be of differing size. Although the MPC8240 multiplexes the row address, column address, and logical bank select bits onto a shared 14-bit memory address bus, individual SDRAM banks may be implemented with memory devices requiring fewer than 28 address bits. The MPC8240 can be configured to provide 12, or 11 row bits to a particular bank, and 10, 9, 8, or 7 column bits, and 2 or 4 logical banks. Table 6-14 summarizes the SDRAM memory configurations supported by the MPC8240.

Note

The largest DRAM that can be supported is limited to 24 total address bits.

System software must configure the MPC8240, using a memory polling algorithm at system start-up, to map correctly the size of each bank in memory. Then, the MPC8240 uses its bank map to assert the appropriate $\overline{CS}[0-7]$ signal for memory accesses according to the provided bank depths. System software must also configure the MPC8240 at system start-up to multiplex appropriately the row and column address bits for each bank. Refer to row-address configuration in the MCCR1. Address multiplexing occurs according to these configuration bits.

Note

Any unused banks should have their starting and ending addresses programmed out of the range of memory banks in use.

If a disabled bank has its starting and ending address defined as overlapping an enabled bank's address space, there may be system memory corruption in the overlapping address range. Always map unused memory banks' starting and ending addresses to memory space that is not used by the system.

Table 6-13 shows the unsupported multiplexed row and column address bits for 32- and 64-bit modes. Configurations using 7 or 8 column address bits in 32-bit data bus mode and 7 column bits in 64-bit data bus mode are not supported as they would create non-contiguous address spaces.

Table 6-13. Unsupported Multiplexed Row and Column Address Bits

32-bit Data Bus Mode	64-bit Data Bus Mode
14x8	—
13x8	—
12x8	—
12x7	12x7
14x7	14x7

Table 6-14. Supported SDRAM Device Configurations—64-bit Mode

SDRAM Devices	Number of Devices in a Physical Bank ¹	Device Configuration	Row Bits x Column Bits x Logical Banks ²	Physical Bank Size ³ (Mbytes)	Eight Physical Banks of Memory (Mbytes)
16 Mbits 2 bank	16	2M x 2 banks x 4 bits	11 x 10 x 2	32	256
	8	1M x 2 banks x 8 bits	11 x 9 x 2	16	128
	4	512K x 2 banks x 16 bits	11 x 8 x 2	8	64
64 Mbits 2 bank	16	8M x 2 banks x 4 bits	13 x 10 x 2	128	1024
	8	4M x 2 banks x 8 bits	13 x 9 x 2	64	512
	4	2M x 2 banks x 16 bits	13 x 8 x 2	32	256
	2	1M x 2 banks x 32 bits	12 x 8 x 2	16	128
64 Mbits 4 bank	16	4M x 4 banks x 4 bits	12 x 10 x 4	128	1024
	8	2M x 4 banks x 8 bits	12 x 9 x 4	64	512
	4	1M x 4 banks x 16 bits	12 x 8 x 4	32	256
	2	512K x 4 banks x 32 bits	11 x 8 x 4	16	128
128 Mbits 2 Banks	8	8M x 2 Banks x 8 bits	13 x 10 x 2	128	1024
	4	4M x 2 Banks x 16 bits	13 x 9 x 2	64	512
	2	2M x 2 Banks x 32 bits	13 x 8 x 2	32	256
128 Mbits 4 Banks	8	4M x 4 Banks x 8 bits	12 x 10 x 4	128	1024
	4	2M x 4 Banks x 16 bits	12 x 9 x 4	64	512
	2	1M x 4 Banks x 32 bits	12 x 8 x 4	32	256

¹ A physical bank is defined for the MPC8240 as a portion of memory addressed through an SDRAM chip select. Certain modules of SDRAM may have two physical banks and requires two chip selects to be programmed to support a single module.

² A logical bank is defined for the MPC8240 as a portion of memory addressed through an SDRAM bank select.

³ The physical bank size is the amount of memory addressed by a single SDRAM chip select.

6.4.2 SDRAM Address Multiplexing

This section describes how the MPC8240 translates processor addresses into SDRAM memory addresses.

Note

The MPC8240 SDRAM memory address signals SDMA[12–0] are labeled with SDMA12 as the most-significant bit (msb) and SDMA0 as the least-significant bit (lsb).

Most SDRAM devices are labeled with A0 as the least significant address input. Therefore, the MPC8240 SDMA[12–0] signals should be connected to SDRAM devices according to Table 6-2.

Figure 6-22 shows the multiplexing of the internal physical addresses $A[0_{msb}-31_{lsb}]$ through SDBA[1-0] and SDMA[12–0] during the row and column phases of the 32-bit mode.

Row x Col x Bank		Physical Address																													
		msb																													lsb
		0–5	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			
11x10x2	SDRAS					BA 0	10	9	8	7	6	5	4	3	2	1	0														
	SDCAS					9	BA 0											8	7	6	5	4	3	2	1	0					
11x9x2	SDRAS					BA 0	10	9	8	7	6	5	4	3	2	1	0														
	SDCAS					BA 0											8	7	6	5	4	3	2	1	0						
13x10x2	SDRAS				12	11	BA 0	10	9	8	7	6	5	4	3	2	1	0													
	SDCAS				9	BA 0											8	7	6	5	4	3	2	1	0						
13x9x2	SDRAS				12	11	BA 0	10	9	8	7	6	5	4	3	2	1	0													
	SDCAS					BA 0											8	7	6	5	4	3	2	1	0						
12x10x4	SDRAS				11	BA 1	BA 0	10	9	8	7	6	5	4	3	2	1	0													
	SDCAS				9	BA 1	BA 0										8	7	6	5	4	3	2	1	0						
12x9x4	SDRAS				11	BA 1	BA 0	10	9	8	7	6	5	4	3	2	1	0													
	SDCAS					BA 1	BA 0										8	7	6	5	4	3	2	1	0						

Figure 6-22. SDRAM Address Multiplexing SDBA[1–0] and SDMA[12–0]—32-Bit Mode

Figure 6-23 shows the multiplexing of the internal physical addresses $A[0_{msb}-31_{lsb}]$ through SDBA[1–0] and SDMA[12–0] during the row and column phases of the 64-bit mode. The shaded cells in Figure 6-23 are the unspecified bits.

SDRAM Interface Operation

Row x Col x Bank		Physical Address																													
		msb																													lsb
		0-4	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		
11x10x2	SDRAS						BA0	10	9	8	7	6	5	4	3	2	1	0													
	SDCAS				9	8	BA0												7	6	5	4	3	2	1	0					
11x9x2	SDRAS						BA0	10	9	8	7	6	5	4	3	2	1	0													
	SDCAS					8	BA0												7	6	5	4	3	2	1	0					
11x8x2	SDRAS						BA0	10	9	8	7	6	5	4	3	2	1	0													
	SDCAS						BA0												7	6	5	4	3	2	1	0					
13x10x2	SDRAS				12	11	BA0	10	9	8	7	6	5	4	3	2	1	0													
	SDCAS		9	8			BA0												7	6	5	4	3	2	1	0					
13x9x2	SDRAS				12	11	BA0	10	9	8	7	6	5	4	3	2	1	0													
	SDCAS				8		BA0												7	6	5	4	3	2	1	0					
12x8x2 or 13x8x2	SDRAS				12	11	BA0	10	9	8	7	6	5	4	3	2	1	0													
	SDCAS						BA0												7	6	5	4	3	2	1	0					
12x10x4	SDRAS				11	BA1	BA0	10	9	8	7	6	5	4	3	2	1	0													
	SDCAS		9	8		BA1	BA0												7	6	5	4	3	2	1	0					
12x9x4	SDRAS				11	BA1	BA0	10	9	8	7	6	5	4	3	2	1	0													
	SDCAS				8	BA1	BA0												7	6	5	4	3	2	1	0					
11x8x4 or 12x8x4	SDRAS				11	BA1	BA0	10	9	8	7	6	5	4	3	2	1	0													
	SDCAS					BA1	BA0												7	6	5	4	3	2	1	0					

Figure 6-23. SDRAM Address Multiplexing SDBA[1-0] and SDMA[12-0]—64-Bit Mode

6.4.3 SDRAM Memory Data Interface

The MPC8240 supports several types of data-path buffering for the SDRAM data interface. Table 6-15 and Table 6-16 describe the data path options available for SDRAM. Unspecified bit settings have undefined behavior.

The data-path between the internal processor core bus and the external memory bus for flow-through buffering mode is shown in Figure 6-24. The flow-through mode is the default mode for the MPC8240.

The registered interface is shown in Figure 6-25. The registered buffering mode allows a higher frequency memory interface at the expense of a clock cycle of latency on SDRAM reads.

The in-line ECC buffering mode allows ECC generation and checking between the internal processor core bus and the external SDRAM data bus as shown in Figure 6-26. In-line ECC and parity checking can be enabled or disabled through configuration bits. The ECC syndrome encoding is described in Table 6-17 and Table 6-18. Unspecified configuration register bit settings have undefined behavior.

Table 6-15. Bit Name and Location Cross-listing

Bit Name	Register and Offset	Bit Number in Register
RAM_TYPE	MCCR1 @F0	17
EDO	MCCR2 @F4	16
PCKEN	MCCR1 @F0	16
WRITE_PARITY_CHK_EN	MCCR2 @F4	19
INLRD_PARECC_CHK_EN	MCCR2 @F4	18
INLINE_PAR_NOT_ECC	MCCR2 @F4	20
INLINE	MCCR4 @FC	22
REGISTERED	MCCR4 @FC	20
RMW_PAR	MCCR2 @F4	0
ECC_EN	MCCR2 @F4	17
MEM_PERR_EN	ErrEnR1 @C0	2
MB_ECC_ERR_EN	ErrEnR2 @C4	3

Table 6-16. SDRAM System Configurations

RAM TYPE	EDO	PKEN	WRITEPARITYCHKEN	INLRDPAARCCCHKEN	INLNPEARNOTECC	INLNIE	REGISTERED	RMWPAR	ECCEN	MEMPERREN	MBECCERREN	Description
0	0	0	0	0	0	0	0	0	0	0	0	Flow-through, no ECC or parity
0	0	0	0	0	0	0	1	0	0	0	0	Registered, no ECC or parity
0	0	1	0	0	0	0	0	0	0	1	0	Flow- through parity
0	0	1	0	0	0	0	1	0	0	1	0	Registered parity
0	0	1	0	0	0	0	0	1	0	1	0	Flow- through RMW parity
0	0	1	0	0	0	0	1	1	0	1	0	Registered RMW parity
0	0	0	0	0	0	1	0	0	0	0	0	In-line, no parity
0	0	0	1	1	1	1	0	0	0	1	0	In-line, parity enabled
0	0	0	1	1	1	1	0	1	0	1	0	In-line, RMW parity enabled
0	0	0	1	1	0	1	0	1	0	1	0	In-line, ECC enabled

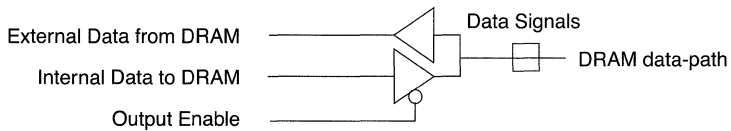


Figure 6-24. SDRAM Flow-Through Memory Interface

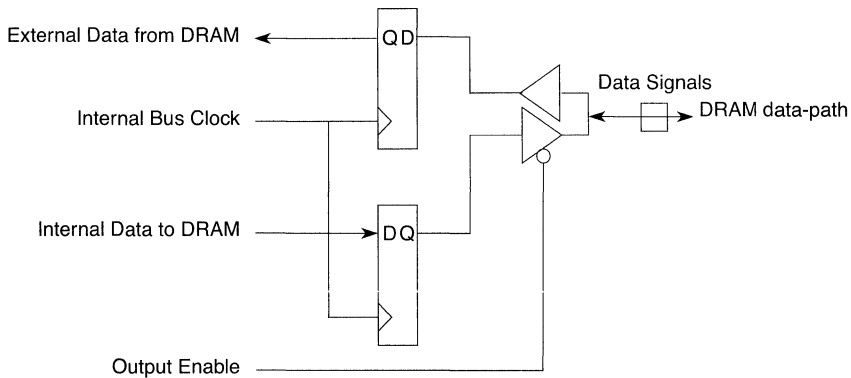


Figure 6-25. SDRAM Registered Memory Interface

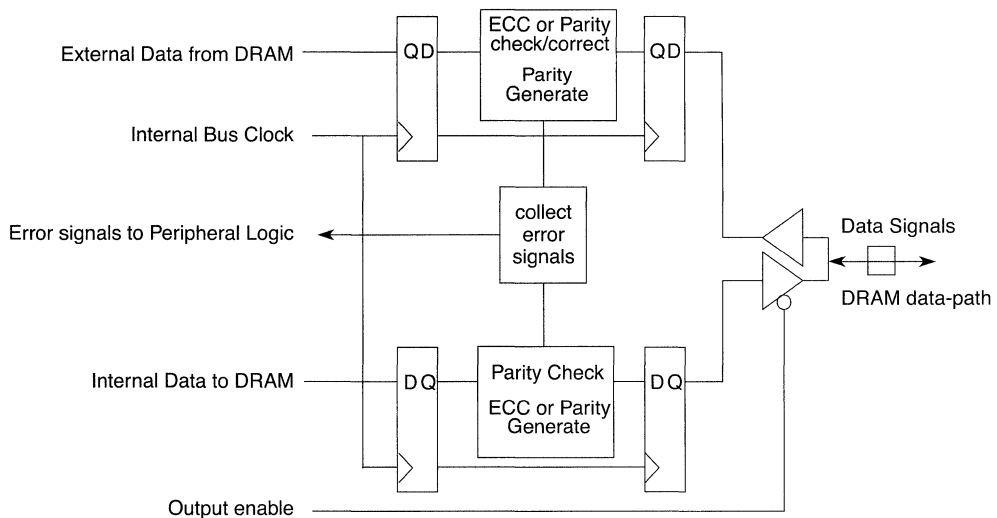


Figure 6-26 SDRAM In-line ECC/Parity Memory Interface

6.4.4 SDRAM ECC

As an alternative to simple parity, the MPC8240 supports ECC for the data-path between the MPC8240 and system memory. ECC not only allows the MPC8240 to detect errors in the memory data-path, it allows the MPC8240 to correct single-bit errors in the 64-bit data-path.

Note

ECC is not supported for systems using a 32-bit data bus.

SDRAM Interface Operation

The in-line ECC logic in the MPC8240 detects and corrects all single-bit errors and detects all double-bit errors and all errors within a nibble. Other errors may be detected but are not guaranteed to be detected or corrected. Multiple bit errors are always reported if detected. However, when a single-bit error occurs, the “ECC Single Bit Error Counter Register: 1, and its value is compared to the “ECC Single Bit Error Trigger Register: 1 byte @ <0xB9>.” If the values are not equal, no error is reported; if the values are equal, then an error is reported. Thus, the single-bit error registers may be programmed so that minor faults with memory are corrected and ignored, but a catastrophic memory failure generates an interrupt. The syndrome equations for the ECC code are shown in Table 6-11 and Table 6-17.

The MPC8240 supports concurrent ECC for the memory data-path and parity for the local processor data-path. ECC and parity may be independently enabled or disabled. The eight signals used for ECC (PAR[0–7]) are also used for local processor parity. The MPC8240 checks ECC on 64-bit memory reads.

The MPC8240 supports a read-modify-write mode similar to the RMW parity mode described in Section 6.3.8, “FPM or EDO DRAM Parity and RMW Parity,” to perform sub-doubleword write operations.

The MPC8240 supports burst-of-four, double-word data beats for accesses to system memory. The burst is always sequential, and the critical double word is always supplied first. For example, if the local processor requests the third double word of a cache block, the MPC8240 reads double words from memory in the order 2-3-0-1.

6.4.6 SDRAM Page Mode Retention

Under certain conditions, the MPC8240 retains four active SDRAM pages for burst or single-beat accesses. These conditions are as follows:

- A pending transaction (read or write) hits one of the currently active internal pages.
- There are no pending refreshes.
- The burst-to-precharge interval (controlled by BSTOPRE[0–9]) has not been exceeded.
- The maximum activate-to-precharge interval (controlled by PGMAX) has not been exceeded.

Note

The BSTOPRE[0–9] parameter is composed of BSTOPRE[0–1] (bits 19–18 of MCCR4), BSTOPRE[2–5] (bits 31–28 of MCCR3), and BSTOPRE[6–9] (bits 3–0 of MCCR4).

Page mode can dramatically reduce access latencies for page hits. Depending on the memory system design and timing parameters, using page mode can save clock cycles from subsequent burst accesses that hit in an active page. SDRAM page mode is controlled by the BSTOPRE[0–9], and PGMAX parameters. Page mode is disabled by clearing the PGMAX or BSTOPRE[0–9] parameters.

The page open duration counter is loaded with BSTOPRE[0–9] every time the page is accessed (including page hits). When the counter expires (or when PGMAX expires) the open page is closed with a precharge bank command. Page hits can occur at any time in the interval specified by BSTOPRE.

The 1 byte memory page mode register at address offset 0xA3 contains the PGMAX parameter that controls how long the MPC8240 retains the currently accessed page (row) in memory. The PGMAX parameter specifies the activate-to-precharge interval (sometimes called row active time or t_{RAS}). The PGMAX value is multiplied by 64 to generate the actual number of clock cycles for the interval. When PGMAX is programmed to 0x00, page mode is disabled.

The value for PGMAX depends on the specific SDRAM devices used, the ROM system, and the operating frequency of the MPC8240. When the interval specified by PGMAX expires, the MPC8240 must close the active page by issuing a precharge bank command. PGMAX must be sufficiently less than the maximum row active time for the SDRAM device to ensure that the issuing of a precharge command is not stalled by a memory access.

When PGMAX expires during a memory access, the MPC8240 must wait for the access to complete before issuing the precharge command to the SDRAM. In the worst case, the MPC8240 initiates a memory access one clock cycle before PGMAX expires. If ROM is located on the memory bus, the longest access that could potentially stall a precharge is a burst read from ROM. If ROM is located on the PCI bus, the longest memory access is a burst read from the SDRAM.

The MPC8240 also requires two clock cycles to issue a precharge bank command to the SDRAM device so the PGMAX interval must be further reduced by two clock cycles. Therefore, PGMAX should be programmed according to the following equation:

$$\text{PGMAX} < [t_{\text{RAS}(\text{MAX})} - (\text{worst case memory access}) - 2] / 64$$

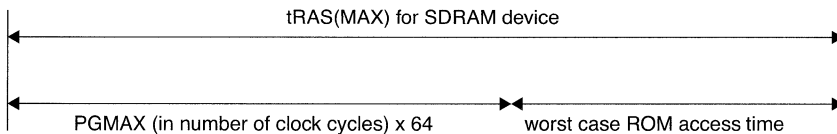


Figure 6-27. PGMAX Parameter Setting for SDRAM Interface

For example, consider a system with a memory bus clock frequency of 66 MHz using SDRAMs with a maximum row active time ($t_{\text{RAS}(\text{MAX})}$) of 100 μs . The maximum number of clock cycles between activate bank and precharge bank commands is 66 MHz x 100 μs = 6600 clock cycles.

If the system uses 8-bit ROMs on the memory bus, a burst read from ROM follows the timing shown in Figure 6-47. Also affecting the ROM access time is MCCR2[TS_WAIT_TIMER]. The minimum time allowed for ROM devices to enter high impedance is two clock cycles. TS_WAIT_TIMER adds clocks ($n-1$) to the minimum disable time. This delay is enforced after all ROM accesses preventing any other memory access from starting. Therefore a burst read from an 8-bit ROM takes

$$\{[(\text{ROMFAL} + 2) \times 8 + 3] \times 4 + 5\} + [2 + (\text{TS_WAIT_TIMER} - 1)] \text{ clock cycles}$$

So, if MCCR1[ROMFAL] = 4 and MCCR2[TS_WAIT_TIMER] = 3, the interval for a local processor burst read from an 8-bit ROM takes

$$\{[(4 + 2) \times 8 + 3] \times 4 + 5\} + [2 + (3 - 1)] = 209 + 4 = 213 \text{ clock cycles.}$$

Plugging the values into the PGMAX equation above,

$$\text{PGMAX} < (6600 - 213 - 2) \div 64 = 99.7 \text{ clock cycles.}$$

The value stored in PGMAX would be 0b0110_0011 (or 99 clock cycles).

6.4.6.1 SDRAM Paging in Sleep Mode

Systems attempting to go to sleep with SDRAM paging enabled must ensure that the following sequence of events occurs in software before the processor core enters sleep mode

- Disable paging by writing 0x00 to the PGMAX register, MPMR @ 0xA3.
- Wait for any open pages to close by allowing a SDRAM refresh interval to elapse; REFINT, MCCR @ 0xF4 bits (15–2)
- Processor core enters Sleep Mode.

Upon waking from sleep, software must perform the following sequence to re-enable paging (if so desired).

- Awake from sleep
- Enable paging by writing the appropriate maximum page open interval based upon the system design to the PGMAX register, MPMR @0xA3. (optional)

6.4.7 SDRAM Power on Initialization

At system reset, boot code must clear MPC8240 configuration bit RAMTYP if the system contains SDRAM. Additionally, all other MPC8240 configuration registers relevant to SDRAM must be initialized. These register fields are as follows:

- RAMTYP—SDRAM, FPM or EDO
- Memory bank start and end addresses
- Memory bank enables
- Row address bit count for each bank
- PCKEN—parity check enable
- SREN—self refresh enable
- REGISTERED—set to indicate clocked memory buffer in data-path
- INLINE—set to indicate in-line parity or ECC in read and write paths
- INLRD_PARECC_CHK_EN—enable in-line read path ECC or parity error reporting
- WRITE_PARITY_CHK_EN—enable write path parity error reporting
- INLINE_PAR_NOT_ECC select between ECC or parity on the memory bus for inline buffer mode only.
- REFINT—interval between refreshes
- RDLAT—column command to internal processor core bus read data latency
- REFREC—refresh recovery interval from last refresh clock cycle to activate command
- ACTORW—activate (command) to read or write (command) interval

- **SDMODE**—mode register data to be transferred to SDRAM array by the MPC8240 —specifies CAS latency, wrap type, and burst length
- **ACTOPRE**—activate (command) to precharge (command) interval
- **PRETOACT**—precharge (command) to activate (command) interval
- **BSTOPRE**—burst to precharge (command) interval (page open interval)
- **RSV_PG**—reserve a page register (only allow 3 simultaneous pages)

Once configuration of all above parameters is complete, the system software must set the the MPC8240 configuration bit **MEMGO**. The MPC8240 then conducts an initialization sequence to prepare the SDRAM array for accesses. The initialization sequence for JEDEC compliant SDRAM is as follows:

- Precharge both internal banks of the SDRAM device.
- Issue 8 refresh commands.
- Issue mode register set command to initialize the mode register.

The SDRAM will then be available for system accesses.

6.4.8 MPC8240 Interface Functionality for JEDEC SDRAMs

All read or write accesses to SDRAM are performed by the MPC8240 using various combinations of the JEDEC standard SDRAM interface commands. SDRAM samples command and data inputs on rising edges of the memory clock. Additionally, SDRAM output data must be sampled on rising edges of the memory clock. Table 6-19 describes the MPC8240 SDRAM interface command and data inputs.

The following SDRAM interface commands are provided by the MPC8240.

- **Bank 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 bank activate is done.
- **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 be done if the row address will change on next 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, additional data will be output without additional read commands. The amount of data so transferred is determined by the burst size.
- **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, additional data is transferred to the sense amplifiers from the data pins without additional write commands. The amount of data so transferred is determined by the burst size. Sub-burst write operations are controlled with **DQM[0–7]**.

SDRAM Interface Operation

- Refresh (similar to CAS before RAS)—Causes a row to be read in both memory 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. Before execution of refresh, all memory banks must be in a precharged state.
- Mode register set (for configuration)—Allows setting of SDRAM options. These options are CAS latency, burst type, and burst length.
 - CAS latency may be chosen as provided by the preferred SDRAM. (Some SDRAMs provide CAS latency 1, 2, 3, some provide CAS latency 1, 2, 3, 4).
 - Burst type must be set to sequential.
 - Although some SDRAMs provide variable burst lengths of 1, 2, 4, 8 page size, the MPC8240 supports only a burst length of 4 or 8. Burst length 4 must be selected for operation with a 64 bit memory interface and 8-beat burst lengths are used with a 32-bit memory interface. Burst lengths of 1 and 2 page size are not supported by the MPC8240. This command is performed by the MPC8240 during system initialization.

The mode register data (CAS latency, burst length and burst type), is provided by software at RESET in the MPC8240 configuration register and is subsequently transferred to the SDRAM array by the MPC8240 after MEMGO is enabled.
- Self refresh (for long periods of standby)—Used when the device will be in standby for very long periods of time. Automatically generates internal refresh cycles to keep the data in both memory banks refreshed. Before execution of this command, all memory banks must be in a precharged state.

Table 6-19. MPC8240 SDRAM Interface Command and Data Inputs

Command	SDRAS	SDCAS	WE	CS	CKE
Bank activate	0	1	1	0	1
Precharge	0	1	0	0	1
Read	1	0	1	0	1
Write	1	0	0	0	1
CBR refresh	0	0	1	0	1
Mode register set	0	0	0	0	1
Self refresh	0	0	1	0	0

The MPC8240 automatically issues a precharge command to the SDRAM when the BSTOPRE or PGMAX intervals have expired, regardless of pending memory transactions from the PCI bus or local processor. The MPC8240 can perform precharge cycles concurrent with snoop broadcasts for PCI transactions.

6.4.9 SDRAM Interface Timing

The MPC8240 supports four-beat bursts to SDRAM.

Note

For single-beat reads, the extra data is ignored and the SDRAM output drivers are disabled using DQM[0–7]. These read bursts are not terminated early if some of the data is irrelevant and a second read or write is pending.

For single-beat writes, the MPC8240 protects non-target addresses by driving the DQM[0–7] signals high. These write bursts are not terminated early if a second write or read is pending.

To accommodate available memory technology across a wide spectrum of operating frequencies, the MPC8240 allows the following SDRAM interface timing intervals to be programmable with granularity of 1 memory clock cycle:

- RDLAT—internal processor core bus data latency from read command
- REFREC—refresh command to activate command interval
- ACTORW—activate command to read or write command interval
- ACTOPRE—activate command to precharge command interval
- PRETOACT—precharge command to activate command interval
- BSTOPRE—burst to precharge command interval (page open interval)

Table 6-20. SDRAM Interface Timing Intervals

Timing Intervals	Definition
RDLAT	The number of clock cycles from the read column command until the first data beat is available on the internal processor core data bus. In normal (registered buffer) mode RDLAT will be 1 cycle longer than the programmed SDRAM CAS latency. In in-line ECC or parity mode, RDLAT is 2 cycles longer than CAS latency. The value of RDLAT should be additionally increased by one if MCCR3[REGDIMM] = 1. Thus, RDLAT = SDRAM CAS latency + buffer mode delay + delay due to REGDIMM value.
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 minimum refresh to activate interval in nanoseconds.
ACTORW	The number of clock cycles from an activate command until a read or write command is allowed. This interval will be listed (nS) in the AC specifications of the user's SDRAM.
ACTOPRE	The number of clock cycles from an activate command until a precharge command is allowed. This interval will be listed (nS) in the AC specifications of the user's SDRAM.
PRETOACT	The number of clock cycles from a precharge command until an activate command is allowed. This interval will be listed (nS) in the AC specifications of the user's SDRAM.
BSTOPRE	The number of clock cycles to maintain a page open after an access. A subsequent access can generate a page hit during this interval. A page hit reloads the BSTOPRE counter. When the interval expires, a precharge is issued to the page.

The value of the above six parameters, (in whole clock cycles) must be set by boot code at system start-up and kept in the MPC8240 configuration register space.

The following figures show SDRAM timing for various types of accesses. Figure 6-28 shows a single-beat read operation.

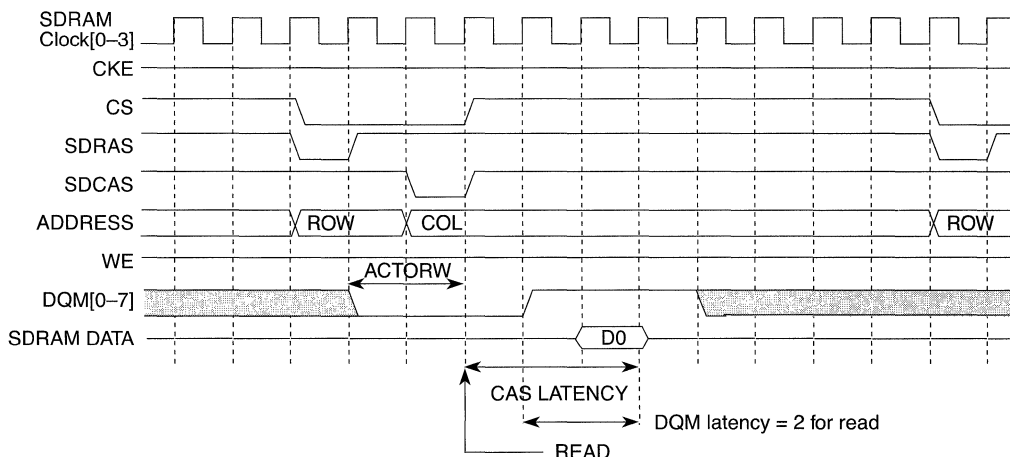


Figure 6-28. SDRAM Single-Beat Read Timing (SDRAM Burst Length = 4)

Figure 6-29 shows a four-beat burst read operation.

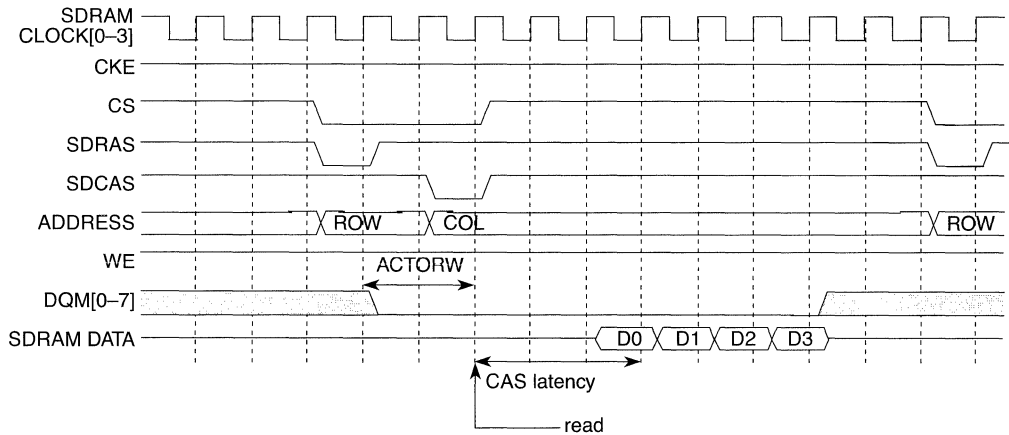


Figure 6-29. SDRAM Four-Beat Burst Read Timing Configuration—64-Bit Mode

Figure 6-30 shows an eight-beat burst read operation.

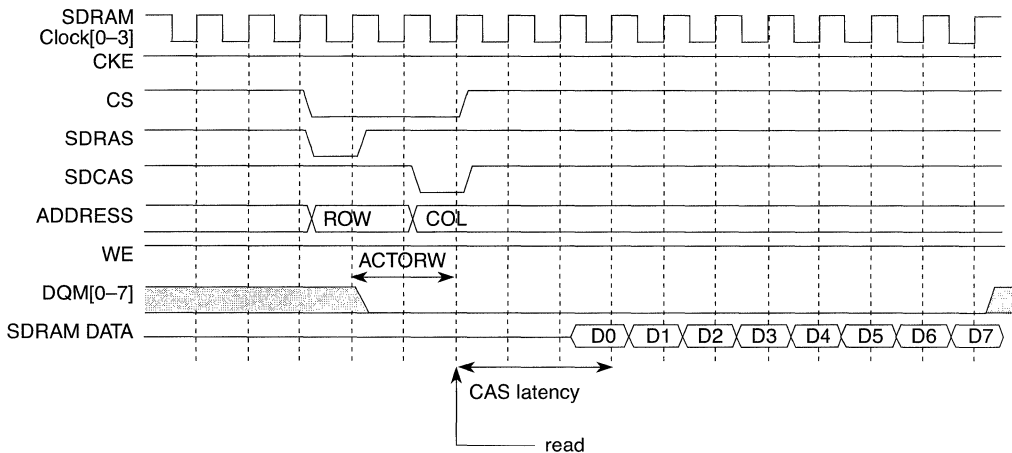


Figure 6-30. SDRAM Eight-Beat Burst Read Timing Configuration—32-Bit Mode

SDRAM Interface Operation

Figure 6-31 shows a single-beat write operation.

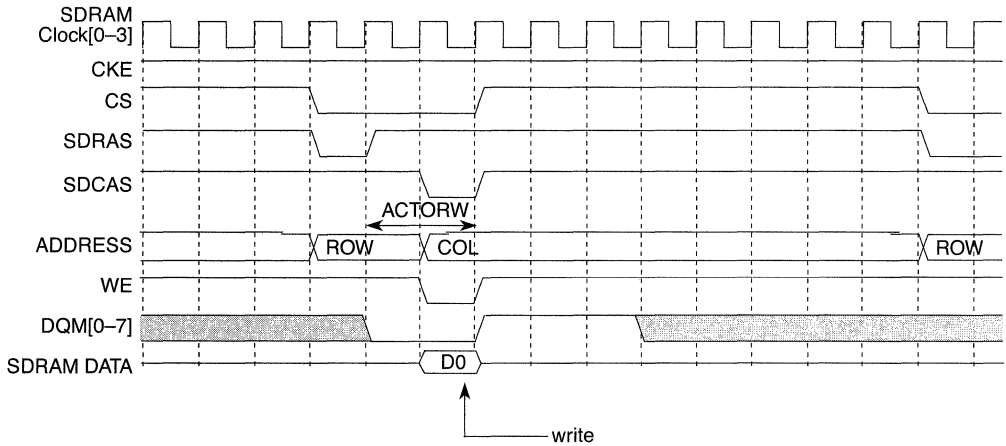


Figure 6-31. SDRAM Single Beat Write Timing (SDRAM Burst Length = 4)

Figure 6-32 shows a four-beat burst-write operation.

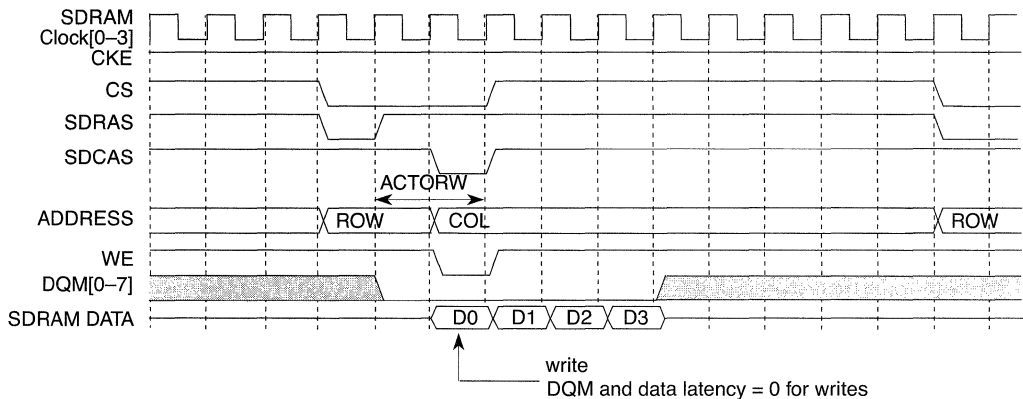


Figure 6-32. SDRAM Four-Beat Burst Write Timing—64-Bit Mode

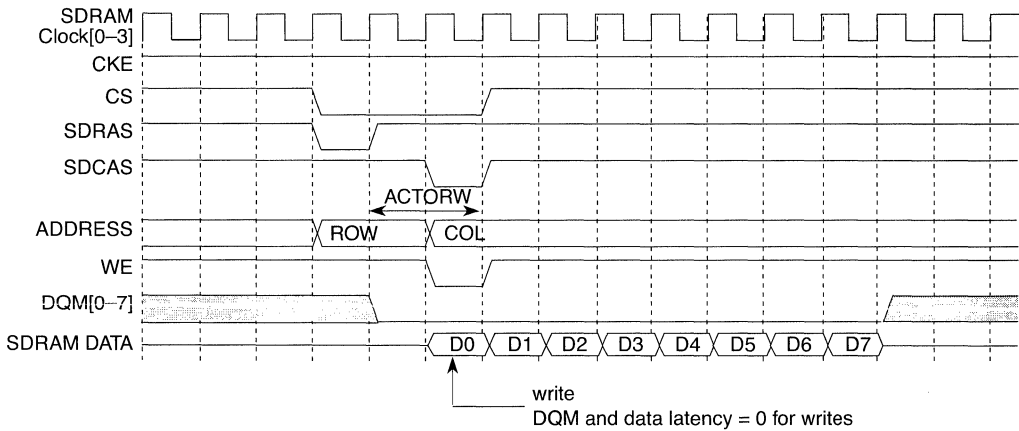


Figure 6-33. SDRAM Eight-Beat Burst Write Timing—32-Bit Mode

6.4.9.1 SDRAM Mode-Set Command Timing

The MPC8240 transfers the mode register data, (CAS latency, burst length, and burst type) stored in MCCR4[SDMODE] to the SDRAM array by issuing the mode-set command. The timing of the mode-set command is shown in Figure 6-34.

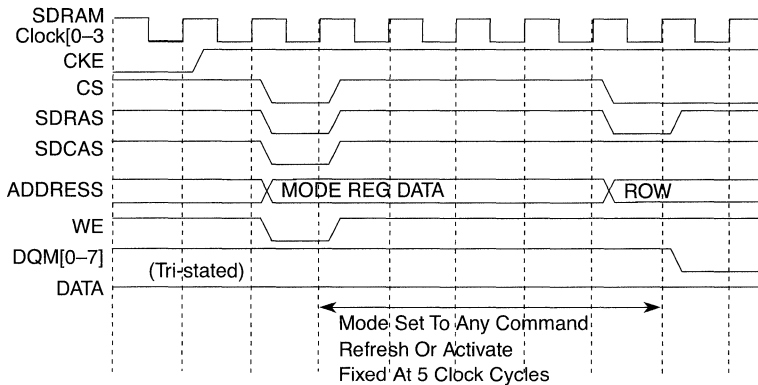


Figure 6-34. SDRAM Mode Register Set Timing

6.4.10 SDRAM Parity and RMW Parity

When configured for SDRAM, the MPC8240 supports two forms of parity checking and generation—normal parity and read-modify-write (RMW) parity. Normal parity assumes that each of the eight parity bits is controlled by a separate DQM signal. Thus, for a single-beat write to system memory, the MPC8240 generates a parity bit for each byte written to memory.

RMW parity assumes that all eight parity bits are controlled by a single DQM signal; therefore, it must be written as a single 8-bit quantity (byte). For any system memory write operations smaller than a double word, the MPC8240 must latch the write data, read a double word (64 bits), check the parity of that double word, merge it with the write data, regenerate parity for the new double word, and finally write the new double word back to memory.

The MPC8240 checks parity on all memory reads, provided parity checking is enabled (PCKEN = 1). The MPC8240 generates parity for the following operations:

- PCI to memory write operations
- Local processor single-beat write operations with RMW parity enabled (RMW_PAR = 1)

The local processor is expected to generate parity for all other memory write operations as the data goes directly to memory and does not pass through the MPC8240.

6.4.10.1 RMW Parity Latency Considerations

When RMW parity is enabled, the time required to read, modify, and write increases latency for both processor single-beat writes and PCI writes to system memory. All other transactions are unaffected and operate as in normal parity mode.

For local processor, single-beat writes to system memory, the MPC8240 latches the data, reads a double-word from system memory (checking parity), and then merges that double-word with the write data from the processor. The MPC8240 then generates new parity bits for the merged double word and writes the data and parity to memory. The read-modify-write process adds six clock cycles to a single-beat write operation. If page mode retention is enabled (BSTOPRE > 0 and PGMAX > 0), the MPC8240 keeps the memory in page-mode for the read-modify-write sequence. Because the processor drives all eight parity bits during burst writes to system memory, these transactions go directly to the SDRAMs with no performance penalty.

For PCI writes to system memory with RMW parity enabled, the MPC8240 latches the data in the internal PCI-to-system-memory-write buffer (PCMWB). If the PCI master writes complete double words to system memory, the MPC8240 generates the parity bits when the PCMWB is flushed to memory. However, if the PCI master writes 32-, 16-, or 8-bit data that cannot be gathered into a complete double word in the PCMWB, a read-modify-write operation is required. The MPC8240 performs a double-word read from system memory (checking parity), and then merges the write data from the PCI master with the data read from memory. The MPC8240 then generates new parity for the merged double word and writes the data and parity to memory. If page mode retention is enabled (BSTOPRE > 0 and PGMAX > 0), the MPC8240 keeps the memory in page mode for the read-modify-write sequence.

6.4.11 SDRAM and In-Line ECC or Parity

The in-line ECC and parity data-path option allows the MPC8240 to detect and correct automatically single bit ECC errors; detect multiple bit ECC errors or parity errors with only one clock cycle penalty on CPU and PCI memory read operations; and generate parity for the internal processor core data bus. For CPU and PCI memory write operations, parity can be checked automatically on the internal processor core data bus and either ECC or parity generated for the memory bus. Table 6-15 and Table 6-16 describe the configuration requirements for this mode.

6.4.12 SDRAM Registered DIMM Mode

The MPC8240 can be configured to support registered DIMMs of SDRAM. To reduce loading, registered DIMMs latch the SDRAM control signals internally before using them to access the array. Enabling the MPC8240's registered DIMM mode (MCCR3 bit 15, REGDIMM = 1) compensates for this delay on the DIMMs control bus by delaying the MPC8240's data and parity buses for SDRAM writes by one additional clock cycle.

Enabling registered DIMM mode has no affect on the bus timing for SDRAM reads or ROM/Flash transfers. However the programmed read latency (RDLAT) time for SDRAM reads must be incremented by one to compensate for the latch delay on the control signals of the registered DIMM. Figure 6-35 shows the registered SDRAM DIMM single beat write timing.

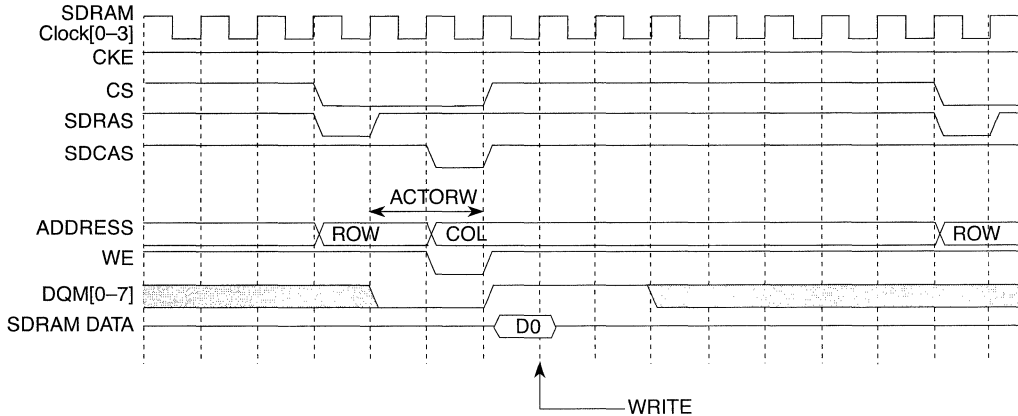


Figure 6-35. Registered SDRAM DIMM Single Beat Write Timing

Figure 6-36 shows the registered SDRAM DIMM burst-write timing.

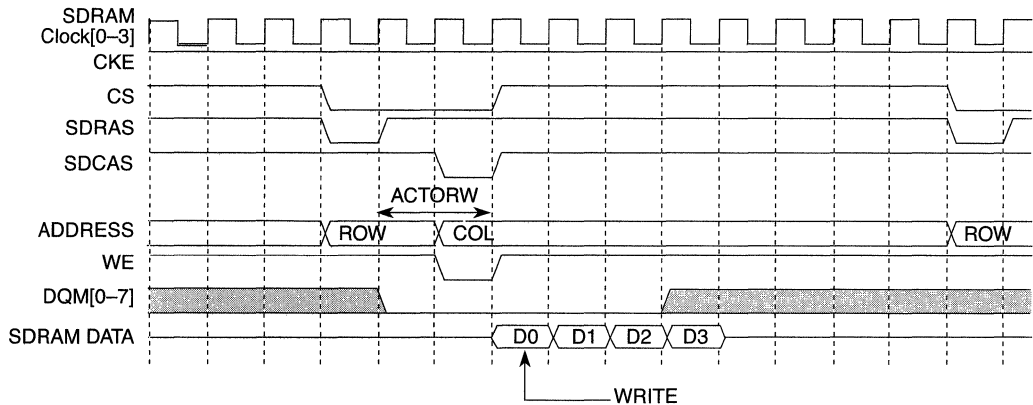


Figure 6-36. Registered SDRAM DIMM Burst-Write Timing

6.4.13 SDRAM Refresh

The memory interface supplies CBR refreshes to SDRAM according to the interval specified in MCCR2[REFINT]. When REFINT expires, the MPC8240 issues a precharge command and then a refresh command to the SDRAM devices. The value stored in REFINT should allow for a potential collision between memory accesses and refresh cycles.

In the worst case, the refresh must wait the number of clock cycles required by the longest access. For example, if ROM is located on the memory bus and a ROM access is in progress at the time a refresh operation needs to be performed, the refresh must wait until the ROM access has completed. If ROM is on the memory bus, the longest access that could potentially stall a refresh is a burst read from ROM.

If ROM is on the PCI bus, the longest memory access is a burst read from the SDRAM. The MPC8240 also has to wait for a precharge command (to close any open pages) before it can issue the refresh command. The MPC8240 requires two clock cycles to issue a precharge to an internal bank; with four pages open simultaneously, this equates to 8 extra clock cycles that must be taken off the refresh interval. Finally, the MPC8240 must wait for the PRETOACT interval to pass before issuing the refresh command.

Therefore, REFINT should be programmed according to the following equation:

$$\text{REFINT} < (\text{per row refresh interval}) - (\text{worst case memory access}) - (\text{PRETOACT}) - 4$$

Consider a typical SDRAM device with a refresh period of 32 mS for a 2K (bank) cycle. This means that it takes 32 mS to refresh each internal bank, and each internal bank has 2,048 rows. It takes 64 mS to refresh the whole SDRAM (two internal banks, 4,096 rows). The refresh time per row is $32 \text{ mS}/2,048 \text{ rows}$ (or $64 \text{ mS}/4,096 \text{ rows}$) = $15.6 \mu\text{S}$. If the memory bus clock is running at 66 MHz, the number of clock cycles per row refresh is $15.6 \mu\text{S} \times 66 \text{ MHz} = 1,030$ clock cycles.

If the system uses 8-bit ROMs on the memory bus, a burst read from ROM follows the timing shown in Figure 6-49. Also affecting the ROM access time is MCCR2[TS_WAIT_TIMER]. The minimum time allowed for ROM devices to enter high impedance is two clock cycles. TS_WAIT_TIMER adds clocks (n-1) to the minimum disable time. This delay is enforced after all ROM accesses preventing any other memory access from starting. Therefore a burst read from an 8-bit ROM will take

$$\{[(\text{ROMFAL} + 2) \times 8 + 3] \times 4 + 5\} + [2 + (\text{TS_WAIT_TIMER} - 1)] \text{ clock cycles}$$

Therefore, if MCCR1[ROMFAL] = 4 and MCCR2[TS_WAIT_TIMER] = 3, the interval for a memory burst read from an 8-bit ROM will take

$$\{[(4 + 2) \times 8 + 3] \times 4 + 5\} + [2 + (3 - 1)] = 209 + 4 = 213 \text{ clock cycles.}$$

Plugging the values into the REFINT equation above,

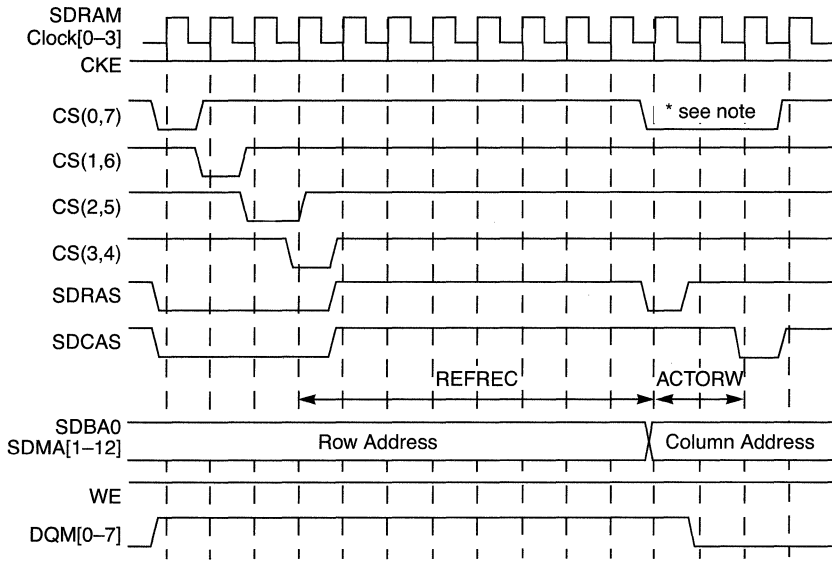
$$\text{REFINT} < 1030 - 213 - 2 - 4 = 811 \text{ clock cycles.}$$

The value stored in REFINT would be 0b00 0011 0010 1010 (or 810 clock cycles).

6.4.13.1 SDRAM Refresh Timing

The CBR refresh timing for SDRAM is controlled by the programmable timing parameter MCCR3[REFREC]. REFREC represents the number of clock cycles from the refresh command until a bank-activate command is allowed. The AC specifications of the specific SDRAM device provides a minimum refresh-to-activate interval.

The MPC8240 implements bank staggering for CBR refreshes, as shown in Figure 6-37. This reduces instantaneous current consumption for memory refresh operations.



Note: Only one CS signal is asserted for the bank-activate and read commands

Figure 6-37. SDRAM Bank Staggered CBR Refresh Timing

6.4.13.2 SDRAM Refresh and Power Saving Modes

The MPC8240's memory interface provides for sleep, doze, and nap power saving modes defined for the local processor architecture. In sleep mode, the MPC8240 can be configured to use the SDRAM self-refresh mode, provide normal refresh to SDRAM, or provide no refresh support. If the MPC8240 is configured to provide no refresh support in Sleep, system software is responsible for appropriately preserving SDRAM data, such as by copying to disk. In doze and nap power saving modes, the MPC8240 supplies normal CBR refresh to SDRAM. Table 6-21 summarizes the MPC8240 configuration bits relevant to power-saving modes.

Table 6-21. SDRAM Controller Power Saving Configurations

Power Saving Mode	Power Save Configuration Bits And Signal Values	Refresh Type	Refresh Configuration Bit Settings
Sleep	PMCR[PM] = 1 PMCR[SLEEP] = 1	Self	PMCR[LP_REF_EN] = 1, MEMCFG[SREN] = 1
		Normal	PMCR[LP_REF_EN] = 1, MEMCFG[SREN] = 0
		None	PMCR[LP_REF_EN] = 0
Nap	PMCR[PM] = 1 PMCR[SLEEP] = 0 PMCR[NAP] = 1	Normal	No additional bits required
Doze	PMCR[PM] = 1 PMCR[SLEEP] = 0, PMCR[NAP] = 0, PMCR[DOZE] = 1	Normal	No additional bits required

Table 6-22 summarizes the refresh types available in each power-saving modes and the relevant configuration parameters.

Table 6-22. SDRAM Power Saving Modes Refresh Configuration

Power Saving Mode	Refresh Type	Power Management Control Register (PMCR)					MCCR1 [SREN]
		PM	DOZE	NAP	SLEEP	LP_REF_EN	
Doze	Normal	1	1	0	0	—	—
Nap	Normal	1	—	1	0	—	—
Sleep	Self	1	—	—	1	1	1
	Normal	1	—	—	1	1	0

SDRAM Interface Operation

The entry timing for self-refreshing SDRAMs is shown in Figure 6-38.

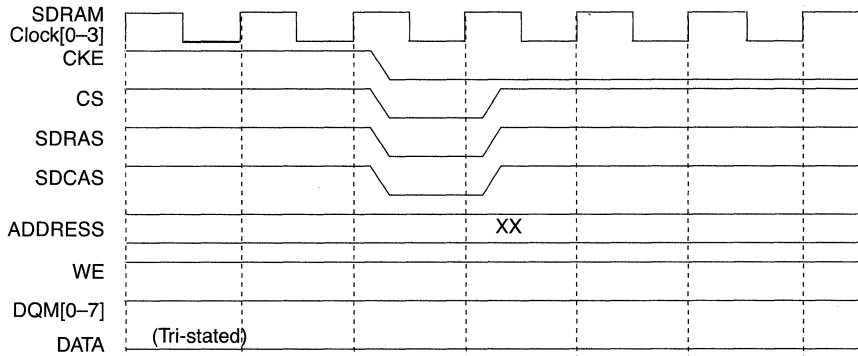


Figure 6-38. SDRAM Self Refresh Entry

The exit timing for self-refreshing SDRAMs is shown in Figure 6-39.

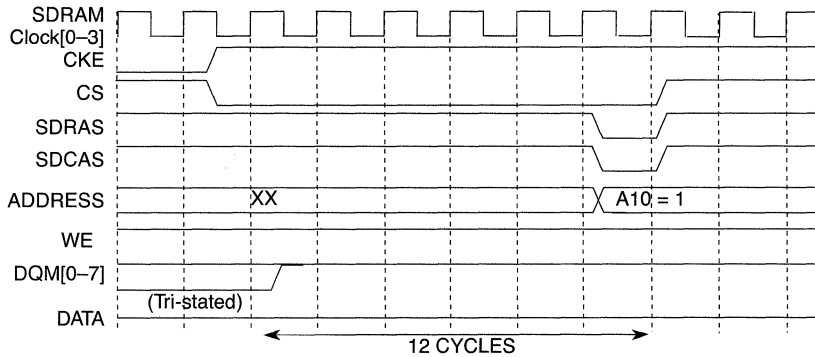


Figure 6-39. SDRAM Self Refresh Exit

6.5 ROM/Flash Interface Operation

For the ROM/Flash interface, the MPC8240 provides 21 address bits, two bank selects, one flash output enable, and one flash write enable.

Figure 6-40 displays a ROM interface block diagram.

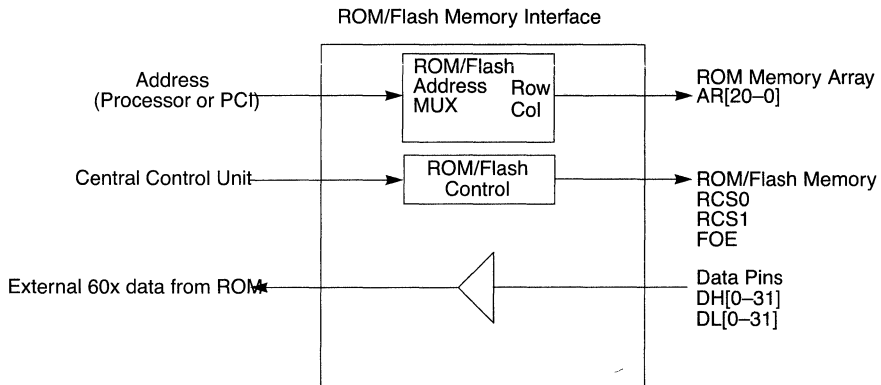
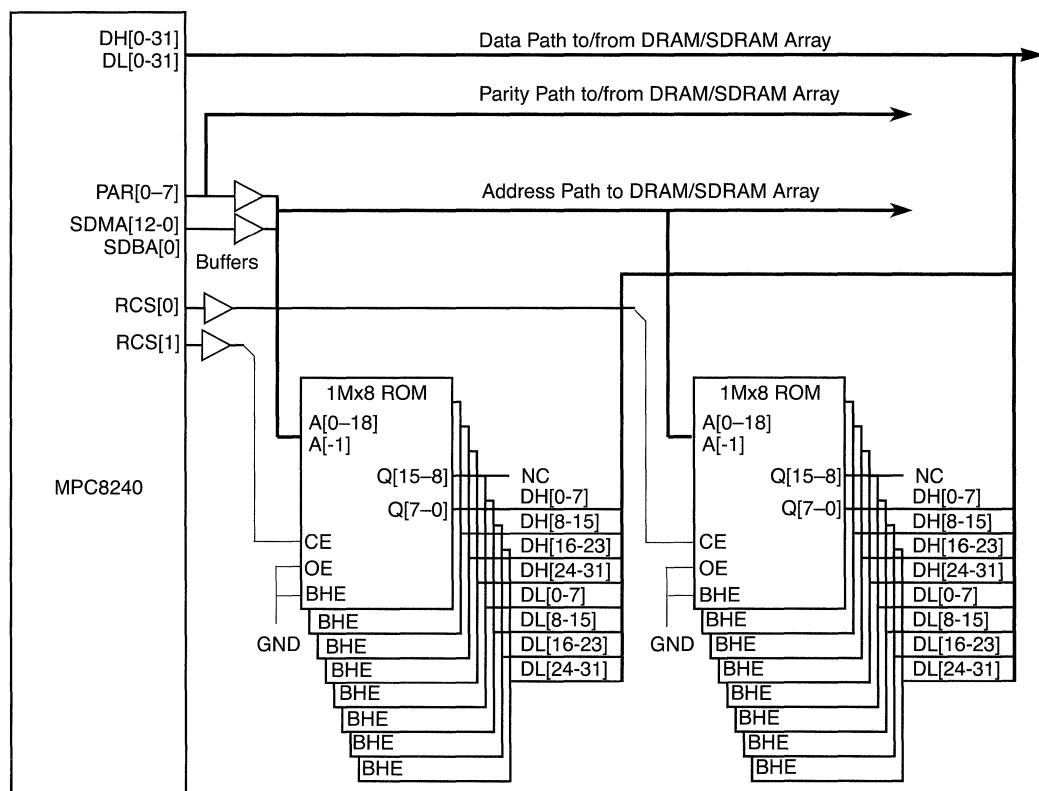


Figure 6-40. ROM Memory Interface Block Diagram

ROM/Flash Interface Operation

Figure 6-41 shows an example of a 16-Mbyte ROM system.



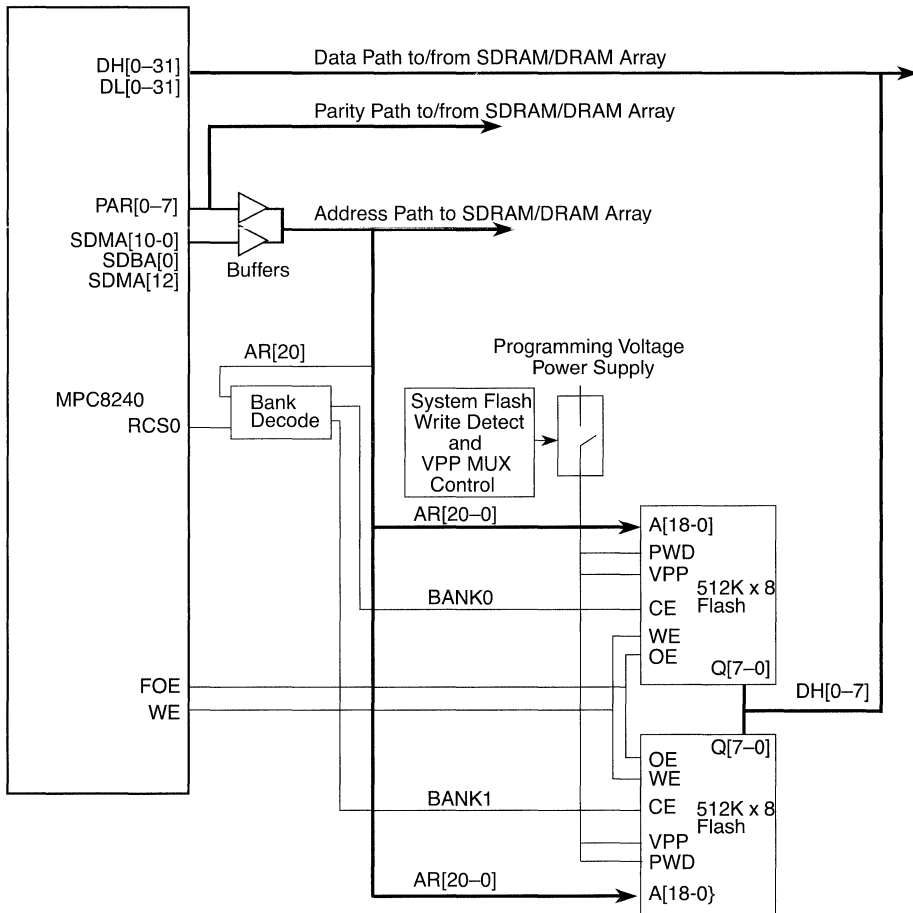
Address Signals (Outputs)	PAR							SDBA	SDMA																			
	0	1	2	3	4	5	6	7	0	10	9	8	7	6	5	4	3	2	1	0								
Logical Names	msb								AR										lsb									
	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0								

Notes:

1. The array of ROM memory devices are 8 Mbit (1M x 8 or 512K x 16) configured for 1M x 8 operation.
2. A[-1] is the lsb of the ROM memory devices.
3. BHE connected to GND enables A[-1] as an input and sets Q[15-8] to Hi-Z.
4. Q[7-0] of the ROM Memory devices are data outputs connected to DH[0-31] and DL[0-31]. DH[0-7] is the most significant byte lane and DL[24-31] is the least significant byte lane.
5. All OE and BHE signals are connected to GND.
6. RCS0 is connected to all CE in Bank 0 (8 Mbytes) and RCS1 is connected to all CE in Bank 1 (8 Mbytes).

Figure 6-41. 16-Mbyte ROM System Including Parity Paths to DRAM—64-Bit Mode

Figure 6-42 shows an example of a 1-Mbyte Flash system.



Address Signals (Outputs)	SDMA	PAR										SDBA	SDMA																											
	12	0	1	2	3	4	5	6	7	0	10	9	8	7	6	5	4	3	2	1	0																			
Logical Names	msb																				AR										lsb									
	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																			

Figure 6-42. 1-Mbyte Flash Memory System Including Parity Paths to DRAM—8-Bit Mode

ROM/Flash Interface Operation

The MPC8240 supports an 8-, 32-, or 64-bit data-path to bank 0. A configuration signal ($\overline{\text{FOE}}$) sampled at reset, determines the bus width of the ROM or Flash device (8-bit, 32-bit, or 64-bit) in bank 0. The data bus width for ROM bank 1 is always 64 or 32 bits as determined by the configuration signal, DL[0], sampled at reset.

Note

The 8-bit data-path requires external decode logic to select between the two devices.

Note

The 21st ROM/Flash address bit (SDMA12/SDBA1) is supported only for bank 0 using the 8-bit data-path and for both banks using the 32-bit data-path.

The extra address bit allows for up to 2 Mbytes of ROM/Flash space in bank 0 for the 8-bit data-path configuration and up to 16 Mbytes of ROM/Flash space using both banks for the 32-bit data-path configuration. For the 64-bit data-path, 20 address bits allow for up to 8 Mbytes per bank for a total of 16 Mbytes using both banks. See Table 6-23.

Table 6-23. Reset Configurations of ROM/Flash Controller

FOE	DL[0]	Bank 0	Bank 1
0	0	32-bit interface 21 address bits 8 Meg space	32-bit interface 21 address bits 8 Meg space
0	1	64-bit interface 20 address bits 8 Meg space	64-bit interface 20 address bits 8 Meg space
1	0	8-bit interface 21 address bits 2 Meg space	32-bit interface 21 address bits 8 Meg space
1	1	8-bit interface 21 address bits 2 Meg space	64-bit interface 20 address bits 8 Meg space

Note

When 64-bit SDRAM/DRAM is selected (DL[0] = 1), ROM/Flash only can be 8- or 64-bit (No 32-bit ROM/Flash).

For systems using the 8-bit interface to bank 0, the ROM/Flash device must be connected to the most-significant byte lane of the data bus DH[0–7]. The MPC8240 performs byte lane alignment for single-byte reads from ROM/Flash memory. The MPC8240 can also perform byte gathering up to 8 bytes for ROM/Flash read operations. The data bytes are gathered and aligned within the MPC8240, and then forwarded to the local processor.

The 16-Mbyte ROM/Flash space is subdivided into two 8-Mbyte banks. Bank 0 (selected by $\overline{\text{RCS0}}$) is addressed from 0xFF80_0000 to 0xFFFF_FFFF. Bank 1 (selected by $\overline{\text{RCS1}}$)

For 32-bit ROMs, the least significant 20 address bits are identical to those previously described for 64-bit ROMs. However, a 21st address bit, SDMA12/SDBA1 (AR[20]), is added as the new most significant address bit. Refer to Table 6-2 on page -5 for “Memory Address Mappings.”

The MPC8240’s two ROM chip select outputs are decoded from the memory address and can be used as bank selects. The MPC8240 can access 16 Mbytes of ROM in systems that have a 64-bit memory bus (8 Mbytes in each bank). In this mode, bank select $\overline{RCS0}$ decodes addresses 0xFF800000–FFFFFFFF, and $\overline{RCS1}$ decodes addresses 0xFF000000–FF7FFFFFFF.

Implementations that require less than 16 Mbytes of ROM may allocate the required ROM to one or both banks. As an example, a 4 Mbyte implementation can place the ROM entirely within the range of $\overline{RCS0}$, (at 0xFFC00000–FFFFFFFF), or can split the ROM between $\overline{RCS1}$ and $\overline{RCS0}$, (at 0xFF600000–FF7FFFFFFF and 0xFFE00000–FFFFFFFF).

The MPC8240 can access 16 Mbytes of ROM in systems that have a 32-bit memory bus (8 Mbytes in each bank). In this mode, bank select $\overline{RCS0}$ decodes addresses 0xFF800000–FFFFFFFF, and $\overline{RCS1}$ decodes addresses 0xFF000000–FF7FFFFFFF. As mentioned previously, implementations that require less than 8 Mbytes of ROM may allocate the required ROM to one or both banks.

The MPC8240 provides programmable access timing for ROM so that systems of various clock frequencies may be implemented. The MPC8240 may also be configured to take advantage of burst (or nibble) mode access time improvements which are available with some ROMs. The programmable parameters for ROM access have granularity of 1 clock cycle, and are named ROMFAL[0–4] and ROMNAL[0–3] in “Memory Control Configuration Register 1: 4 Bytes @ <F0>.”

ROMFAL represents wait states in the access time for non-bursting ROMs, and also measures wait states for the first data beat from bursting ROMs. If ROMFAL[0–4] is programmed to 00000, the default access time is 3 clock cycles for 64 or 32 bit read accesses only and 2 clock cycles for 8 bit read accesses. All write accesses are 2 clock cycles. Any value specified by ROMFAL is added to this default.

ROMNAL represents wait states in access time for nibble (or burst) mode accesses to bursting ROMs. If ROMNAL[0–3] is programmed to 0000, the default, nibble mode access time is 2 clock cycles. Any value specified by ROMNAL is added to this default. To enable the burst mode timing capability, memory configuration register bit “BURST,” in Table 5-37, must be set by boot code.

ROMFAL and ROMNAL will be configured to their maximum value at reset in order to accommodate initial boot code fetches. The “BURST” configuration bit will be cleared at reset. ROM interface timing configuration, and use of the ROMFAL and ROMNAL parameters, is shown in Figure 6-46, Figure 6-47, and Figure 6-48.

ROM/Flash Interface Operation

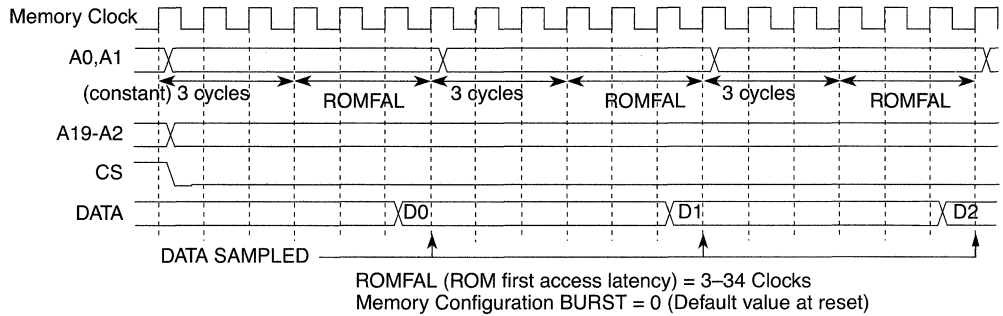


Figure 6-46. Non Burst ROM/Flash Read Access Timing—32- or 64-Bit Mode

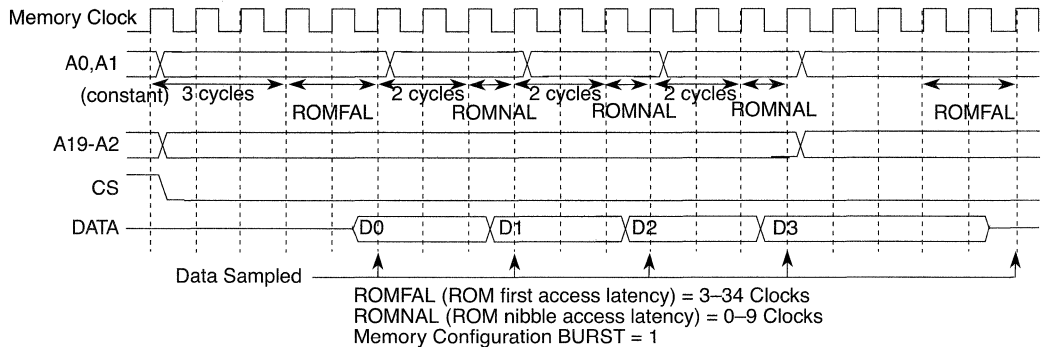


Figure 6-47. Burst ROM/Flash Read Access Timing (Cache Block)—64-Bit Mode

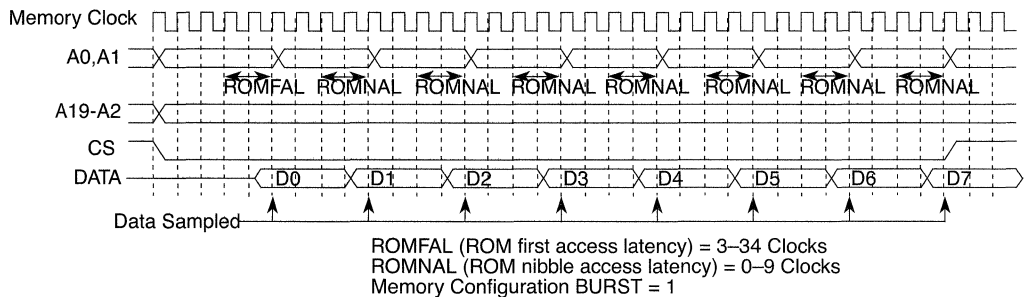


Figure 6-48. Burst ROM/Flash Read Access Timing (Cache Block)—32-Bit Mode

6.5.3 8-Bit ROM/Flash Interface Timing

The MPC8240 provides 21 address bits for accessing 2 Mbytes of external 8-bit ROM/Flash memory. The most significant address bit is SDMA12/SDBA1. The next eight most significant address bits are provided as an alternate function on the MPC8240's parity signals, PAR[0–7] (AR[19–12]). The remaining 12 low order address bits are provided on the MPC8240's SDBA[0] (AR[11]) and SDMA[10–0] (AR[10–0]) signals with SDMA[0] (AR[0]) the least significant bit. Refer to Table 6-2 on page -5 for “Memory Address Signal Mappings.”

The MPC8240 also provides a chip select ($\overline{\text{RCS0}}$), output enable ($\overline{\text{FOE}}$), and write enable ($\overline{\text{WE}}$) to facilitate both read and write accesses to Flash memory. The MPC8240 supports x8 organizations of flash memory up to a total space of 2 Mbytes. Chip select $\overline{\text{RCS0}}$ is decoded from the memory address, and is active for addresses in the range 0xFFE00000–FFFFFFFF for Flash.

The MPC8240 performs byte lane alignment for byte reads from Flash (x8) boot memory. The MPC8240 will gather bytes for half word and double word reads from Flash (x8) boot memory.

The MPC8240 provides programmable timing for read and write access to Flash, with granularity of 1 system clock cycle. ROMFAL[0–4] is used to measure read and write cycle wait states. ROMNAL[0–3] is used to measure write recovery time. Refer to Figure 6-49 on page -70 for further information.

System logic multiplexes high voltage to the Flash memory as required for write operations. This may be done with special logic through I/O space. Refer to Figure 6-50 “8-, 32-, 64-Bit Flash Write Access Timing.”

ROM/Flash Interface Operation

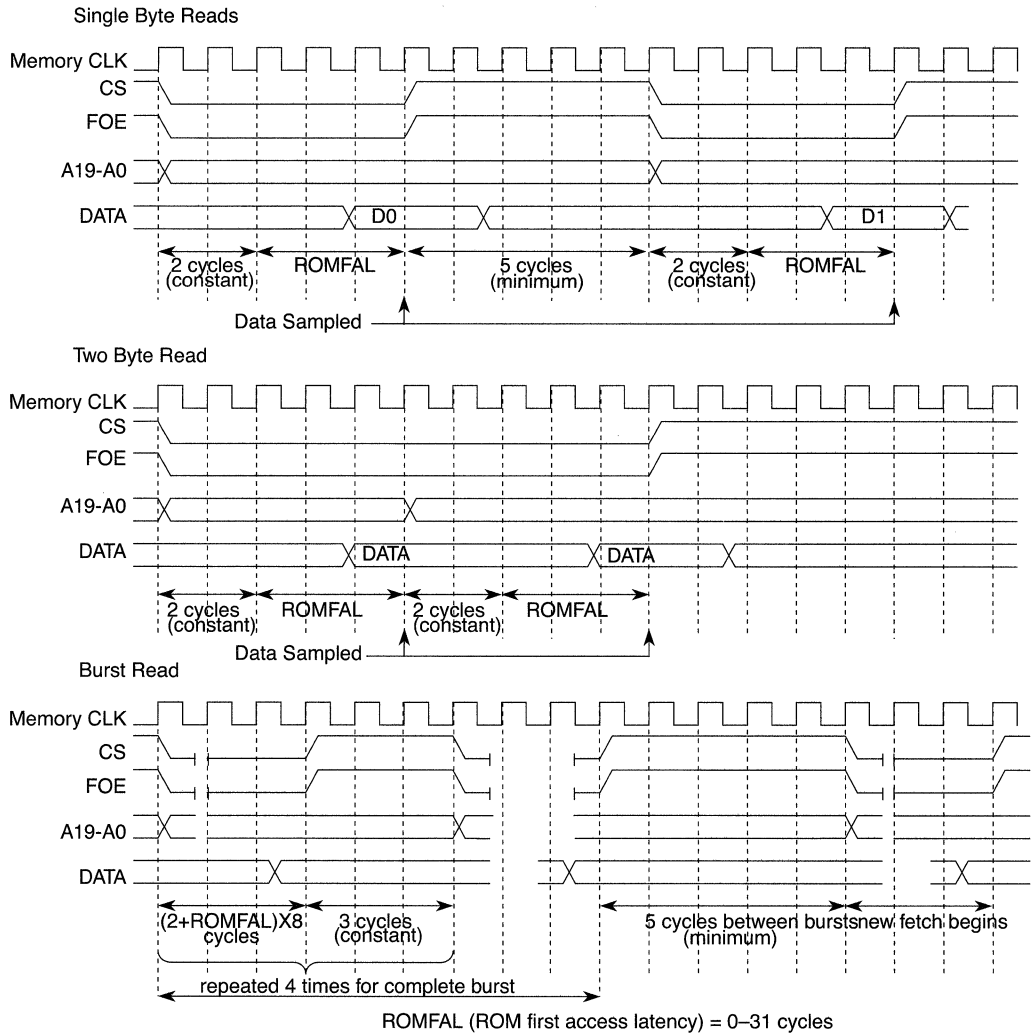


Figure 6-49. 8-Bit ROM/Flash Read Access Timing

6.5.4 ROM/Flash Interface Write Operations

PICR1 [FLASH_WR_EN] must be set for write operations to Flash memory. FLASH_WR_EN controls whether write operations to Flash memory are allowed. FLASH_WR_EN is cleared at reset to disable write operations to Flash memory.

Writes to Flash can be locked out by setting PCIR2 [FLASH_WR_LOCKOUT]. When this bit is set, the MPC8240 disables writing to Flash memory, even if FLASH_WR_EN is set. Once set, the FLASH_WR_LOCKOUT parameter can be cleared only by a hard reset.

If the system attempts to write to read-only devices in a bank, bus contention may occur. This is because the write data is driven onto the data bus when the read-only device is also

trying to drive its data onto the data bus. This situation can be avoided by disabling writes to the system ROM space using `FLASH_WR_EN` or `FLASH_WR_LOCKOUT` or by connecting the Flash output enable (\overline{FOE}) signal to the output enable on the read-only device.

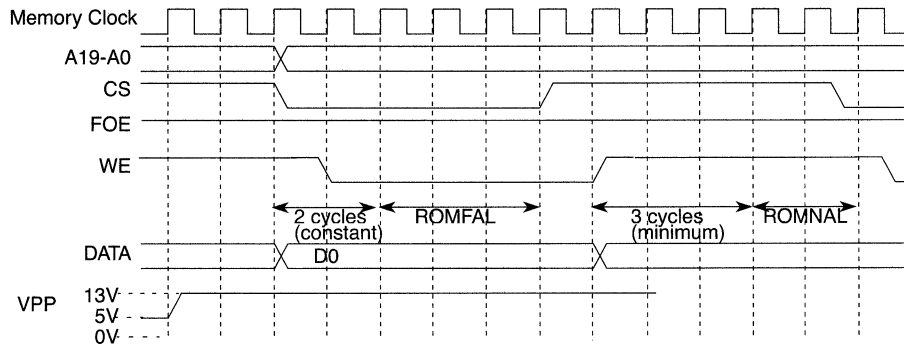
System logic is responsible for multiplexing the required high voltages to the Flash memory for write operations.

The MPC8240 accommodates only single-beat writes to Flash memory. If an attempt is made to write to Flash with a data size other than the full data-path size, the MPC8240 does not report an error. Thus, if software is writing to Flash writes should be the data-path width (8-, 32- or 64-bit) since there is only a single Write Enable (WE) strobe available.

6.5.5 ROM/Flash Interface Write Timing

The parameter `MCCR1[ROMNAL]` controls the Flash memory write recovery time (that is, the number of cycles between write pulse assertions). The actual recovery cycle count is four cycles more than the value specified in `ROMNAL`. For example, when `ROMNAL = 0b0000`, the write recovery time is four clock cycles; when `ROMNAL = 0b0001`, the write recovery time is five clock cycles; when `ROMNAL = 0b0010`, the write recovery time is six clock cycles; and so on. `ROMNAL` is set to the maximum value at reset. To improve performance, initialization software should program a more appropriate value for the device being used.

Figure 6-50 shows the write access timing of the Flash interface.



Vpp multiplexed by system logic with appropriate setup time to write cycle.

Figure 6-50. 8-, 32-, or 64-Bit Flash Write Access Timing

6.5.6 Port X Interface

The MPC8240's memory interface is flexible enough to allow the system designer to connect other non-memory devices to it. This functionality is typically called Port X. By sharing the ROM/Flash interface capabilities with general purpose I/O devices it is possible to configure a wide range of devices with the MPC8240. Figure 6-52 and Figure 6-53 show two examples of Port X implementations. Adding miscellaneous devices to the MPC8240 memory bus limits the total memory devices or maximum bus speed due to signal loading constraints and address space limitations.

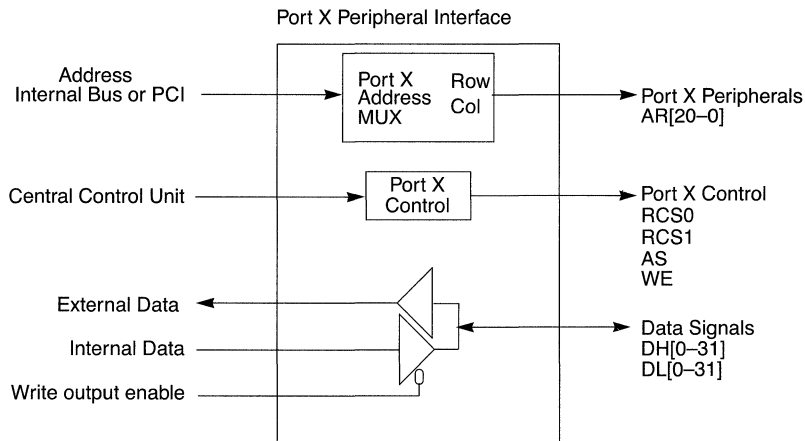


Figure 6-51. Port X Peripheral Interface Block Diagram

Note

Since the MPC8240 shares the Port X interface with the Flash interface data size writes other than the full memory width may be desired (ex. 16-Bit write to a Port X device on a memory interface configured as 64 bit). The MPC8240 allows these transactions and does not report an error.

Data is provided on the data bus as with a memory device. Address and control are provided on the address signals. The \overline{AS} signal's falling and rising edges are programmable to provide a latch strobe or edge reference to allow the external device to latch the data, address, or control signals from the memory interface signals. \overline{AS} is driven active with all accesses to the ROM/Flash address space. This allows for Port X devices to share the address space with ROM devices.

The address space is between 0xFF00_0000–0xFFFF_ FFFF.

The timing of the \overline{AS} signal is controlled by two programmable parameters, ASFALL[0–3] and ASRISE[0–3]. See “Memory Control Configuration Register 2 (DRAM Access Time): 4 Bytes @ <F4>.” ASFALL controls when the \overline{AS} signal transitions from a logic 1 to a logic 0 in relation to the $\overline{RCS}[0–1]$ transition from logic 1 to logic 0. If ASFALL is set to 0b0000, the \overline{AS} is asserted on the same clock cycle as the $\overline{RCS}[0–1]$. A value greater than 0b0000 adds that number of clock cycles to the difference between \overline{AS} and $\overline{RCS}[0–1]$. For example, an ASFALL value of 0b0011 means that the \overline{AS} asserts 3 clock cycles after $\overline{RCS}[0–1]$.

The ASRISE parameter controls when \overline{AS} negates. For example, a ASRISE value of 0b0100 means that the \overline{AS} negates 4 clock cycles after it asserts. Setting ASRISE = 0 effectively disables \overline{AS} and remains negated. At reset, both ASRISE and ASFALL are initialized to 0. This timing is described in Figure 6-54 and Figure 6-52.

Due to restrictions in the ROM and Flash controllers, the ASFALL and ASRISE parameters should be programmed as $ASRISE + ASFALL \leq ROMFAL + 5$ if the ROM interface is programmed to support 8-bit data bus mode for $\overline{RCS}0$, (DBUS_SIZE = x1). Otherwise ASFALL and ASRISE should be programmed as $ASRISE + ASFALL \leq ROMFAL + 6$.

Note

\overline{AS} may not negate between back-to-back Port X transfers if ASFALL is set to 0x0, and ASRISE is set to the maximum allowed value.

The ROM and Flash controllers are capable of multiple-beat read operations. If a Port X device is accessed with a multiple-beat operation, \overline{AS} asserts and negates once after $\overline{RCS}[0–1]$ negate, not multiple times.

Note

There are minimum negation times for $\overline{RCS}[0-1]$ in between Port X transactions:

- 3 clocks (8-bit data bus reads and all writes)
- 4 clocks (32- or 64-bit data bus reads)

ROM/Flash Interface Operation

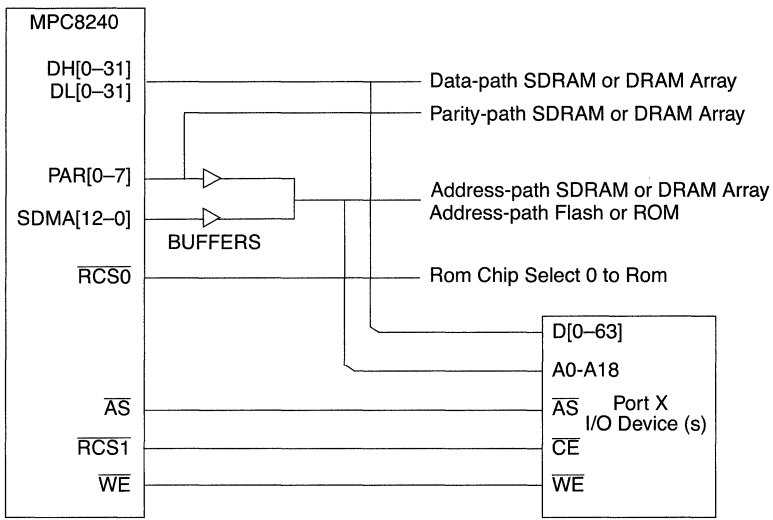


Figure 6-52. Example of Port X Peripheral Connected to the MPC8240

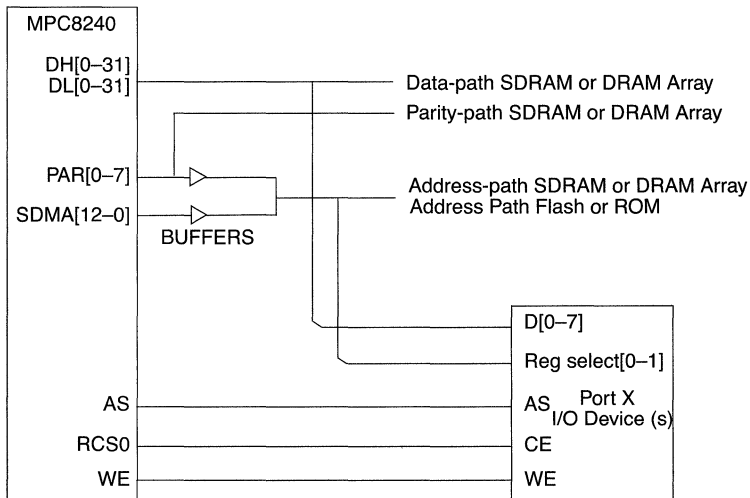


Figure 6-53. Example of Port X Peripheral Connected to the MPC8240

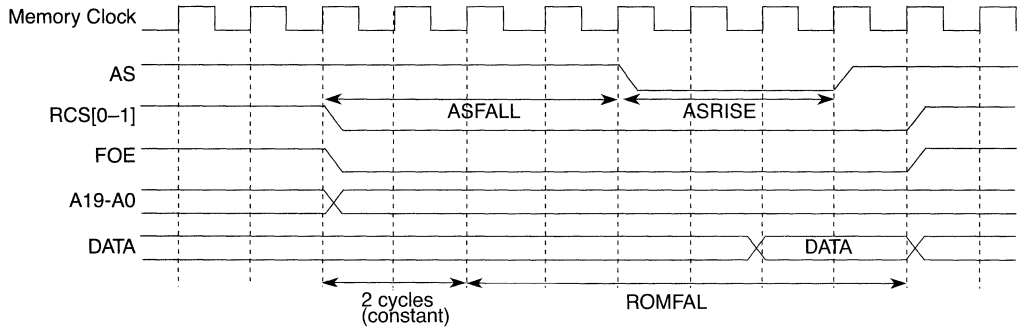


Figure 6-54. Port X Example Read Access Timing

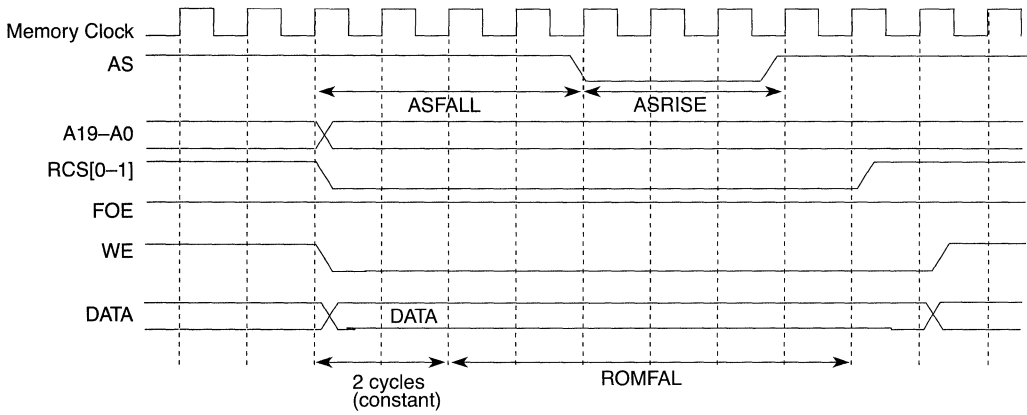


Figure 6-55. Port X Example Write Access Timing

Chapter 7

Central Control Unit

The MPC8240 uses internal buffering to store addresses and data moving through it and to maximize opportunities for concurrent operations. A central control unit directs the flow of transactions through the MPC8240 performing internal arbitration and coordinating the internal and external snooping. This chapter describes the internal buffering and arbitration logic of the MPC8240 central control unit (CCU). See Chapter 2, “PowerPC Processor Core,” for more detailed information on the kinds of internal transactions that are snooped by the processor, the PCI interface, and the memory interface.

7.1 Internal Buffers

For most operations, the data is latched internally in one of eight data buffers. When data transfers between the processor core and system memory, with the exception of snoop copy-backs and burst writes when ECC is enabled, accesses occur directly on the shared data bus, so no internal data buffering is required for those transactions.

Each of the eight data buffers has a corresponding address buffer. An additional buffer stores the address of any processor accesses to system memory. All transactions entering the MPC8240 have their addresses stored in the internal address buffers. The address buffers allow the addresses to be snooped as other transactions attempt to go through the MPC8240. This is especially important for write transactions that enter the MPC8240 because memory can be updated out-of-order with respect to other transactions.

The CCU directs the bus snooping (provided snooping is enabled) for each PCI access to system memory to enforce coherency between the PCI-initiated access and the L1 data cache. All addresses are snooped in the order that they are received from the PCI bus. For systems that do not require hardware-enforced coherency, snooping can be disabled by setting the `CF_NO_SNOOP` parameter in `PICR2`. Note that if snooping is disabled, the PCI exclusive access mechanism (the `LOCK` signal) does not affect the transaction. That is, the transaction will complete, but the processor will not be prohibited from accessing the cache line.

The MPC8240 supports critical word first burst transactions (doubleword aligned) from the processor core. The CCU transfers this doubleword of data first, followed by doublewords from increasing addresses, wrapping back to the beginning of the eight-word block, as required.

Internal Buffers

Figure 7-1 depicts the organization of the internal buffers.

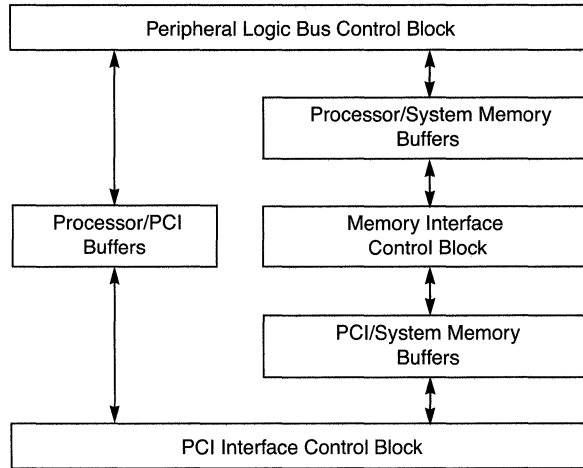


Figure 7-1. MPC8240 Internal Buffer Organization

7.1.1 Processor Core/System Memory Buffers

Because systems using the MPC8240 have a shared data bus between the processor and system memory, for most cases it is unnecessary to buffer data transfers between these devices. However, there is a 32-byte copy-back buffer which is used for temporary storage of L1 copy-back data due to snooping PCI-initiated reads from memory and processor burst writes when ECC is enabled. The copy-back buffer can only be in one of two states—invalid or modified with respect to system memory. Since the buffer is only used for burst write data, the entire cache line in the buffer is always valid if any part of the cache line is valid.

Figure 7-2 shows the address and data buffers between the peripheral logic bus and the system memory bus.

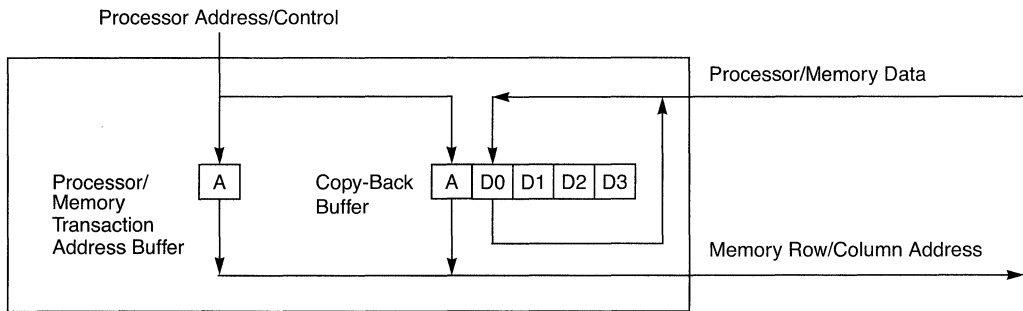


Figure 7-2. Processor/System Memory Buffers

In the case of a snoop for a PCI read from system memory that causes an L1 copy-back, the copy-back data is simultaneously latched in the copy-back buffer and the PCI-read-from-system-memory buffer (PCMRB). Once the L1 copy-back is complete, the data is forwarded to the PCI agent from the PCMRB. The MPC8240 flushes the data in the copy-back buffer to system memory at the earliest available opportunity.

For processor burst writes to memory with ECC enabled, the MPC8240 uses the copy-back buffer as a temporary holding area while it generates the appropriate ECC codes to send to memory.

Once the copy-back buffer has been filled, the data remains in the buffer until the system memory bus is available to flush the copy-back buffer contents to system memory. During the time that modified data waits in the copy-back buffer, all transactions to system memory space are snooped against the copy-back buffer. All PCI-initiated transactions that hit in the copy-back buffer cause the copy-back buffer to have the highest priority for being flushed out to main memory.

7.1.2 Processor/PCI Buffers

There are three data buffers for processor accesses to PCI—one 32-byte processor-to-PCI-read buffer (PRPRB) for processor reads from PCI, and two 16-byte processor-to-PCI-write data buffers (PRPWBs) for processor writes to PCI.

Figure 7-3 shows the address and data buffers between the peripheral logic bus and the PCI bus.

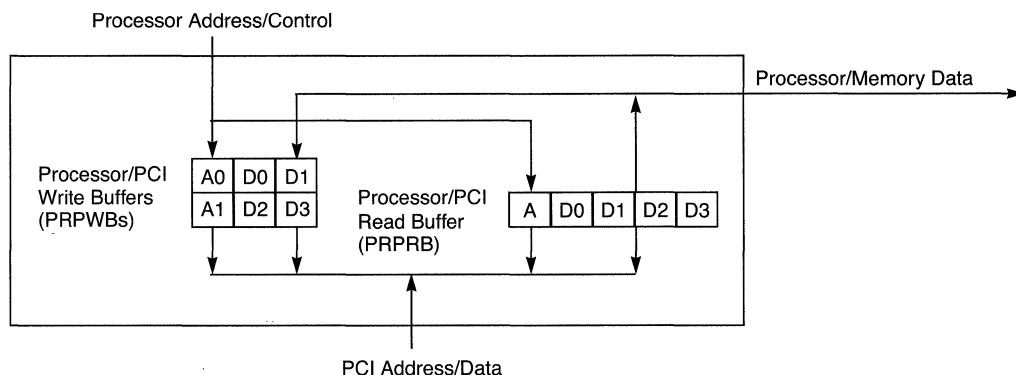


Figure 7-3. Processor/PCI Buffers

7.1.2.1 Processor-to-PCI-Read Buffer (PRPRB)

Processor reads from PCI require buffering for two primary reasons. First, the processor bus uses a critical-word-first protocol, while the PCI bus uses a zero-word-first protocol. The MPC8240 requests the data zero-word-first, latches the requested data, and then delivers the data to the processor core critical-word-first.

The second reason is that if the target for a processor read from PCI disconnects part way through the data transfer, the MPC8240 may have to handle a system memory access from an alternate PCI master before the disconnected transfer can continue.

When the processor requests data from the PCI space, the data received from the PCI is stored in the PRPRB until all requested data has been latched. The CCU does not terminate the address tenure of the internal transaction until all requested data is latched in the PRPRB. If the PCI target disconnects in the middle of the data transfer and an alternate PCI master acquires the bus and initiates a system memory access, the CCU retries the ongoing internal transaction with the processor core so that the incoming PCI transaction can be snooped. A PCI-initiated access to system memory may require a snoop transaction on the internal peripheral logic bus and also a copy-back. The CCU does not provide the data to the peripheral logic bus (for the processor to PCI read transaction) until all outstanding snoops for PCI writes to system memory have completed.

Note

If a processor read from a PCI transaction is waiting for a PCMWB snoop to complete (that is, data has been latched into PRPRB from the PCI bus but has not yet returned to the processor—perhaps the processor must retry the read), all subsequent requests for PCI writes to system memory will be retried on the PCI bus.

The PCI interface of the MPC8240 continues to request the PCI bus until the processor's original request is completed. When the next processor transaction starts, the address is snooped against the address of the previous transaction (in the internal address buffer) to verify that the same data is being requested. Once all the requested data is latched, and all PCI write to system memory snoops have completed, the CCU completes the data transfer to the processor.

For example, if the processor initiates a critical-word-first burst read, starting with the second double word of a cache line, the read on the PCI bus begins with the cache-line-aligned address. If the PCI target disconnects after transferring the first half of the cache line, the MPC8240 re-arbitrates for the PCI bus, and when granted, initiates a new transaction with the address of the third double word of the line. If an alternate PCI master requests data from system memory while the MPC8240 is waiting for the PCI bus grant, the central control unit internally retries the processor core transaction to allow the PCI-initiated transaction to be snooped by the processor core. When the processor snoop is complete, the subsequent processor transaction is compared to the latched address and attributes of the PRPRB to ensure that the processor is requesting the same data. Once all data requested by the processor is latched in the PRPRB, the data is transferred to the processor, completing the transaction.

7.1.2.2 Processor-to-PCI-Write Buffers (PRPWBs)

There are two 16-byte buffers for processor writes to PCI. These buffers can be used together as one 32-byte buffer for processor burst writes to PCI, or separately for single-beat writes to PCI. This allows the MPC8240 to support both burst transactions and streams of single-beat transactions. The MPC8240 performs store gathering (if enabled) of sequential accesses within the 16-byte range that makes up either the first or second half of the cache line. All transfer sizes are gathered if enabled ($\text{PICR1}[\text{ST_GATH_EN}] = 1$).

The internal buffering minimizes the effect of the slower PCI bus on the higher-speed peripheral logic bus that interfaces to the processor core. Once the processor write data is latched internally, the internal peripheral logic bus is available for subsequent transactions without having to wait for the write to the PCI target to complete. Note that both PCI memory and I/O accesses are buffered. Device drivers must take into account that writes to I/O devices on the PCI bus are posted. The processor may believe that the write has completed while the MPC8240 is still trying to acquire mastership of the PCI bus.

If the processor core initiates a burst write to PCI, the processor data transfer is delayed until all previous writes to PCI are completed, and then the burst data from the processor fills the two PRPWBs. The address and transfer attributes are stored in the first address buffer.

For a stream of single-beat writes, the data for the first transaction is latched in the first buffer and the MPC8240 initiates the transaction on the PCI bus. The second single-beat write is then stored in the second buffer. For subsequent single-beat writes, store gathering is possible if the incoming write is to sequential bytes in the same half cache line as the previously latched data. Store gathering is only used for writes to PCI memory space, not for writes to PCI I/O space. The store gathering continues until the buffer is scheduled to be flushed or until the processor issues a synchronizing transaction.

For example, if both PRPWBs are empty and the processor issues a single-beat write to PCI, the data is latched in the first buffer and the PCI interface of the MPC8240 attempts to acquire the PCI bus for the transfer. The data for the next processor-to-PCI write transaction is latched in the second buffer, even if the second transaction's address falls within the same half cache line as the first transaction. While the PCI interface is busy with the first transfer, any sequential processor single-beat writes within the same half cache line as the second transfer are gathered in the second buffer until the PCI bus becomes available.

7.1.3 PCI/System Memory Buffers

There are four data buffers for PCI accesses to system memory—two 32-byte PCI-to-system memory read buffers (PCMRBs) for PCI reads from system memory and two 32-byte PCI-to-system memory write buffers (PCMWBs) for PCI writes to system memory. Figure 7-4 shows the address and data buffers between the PCI bus and the system memory.

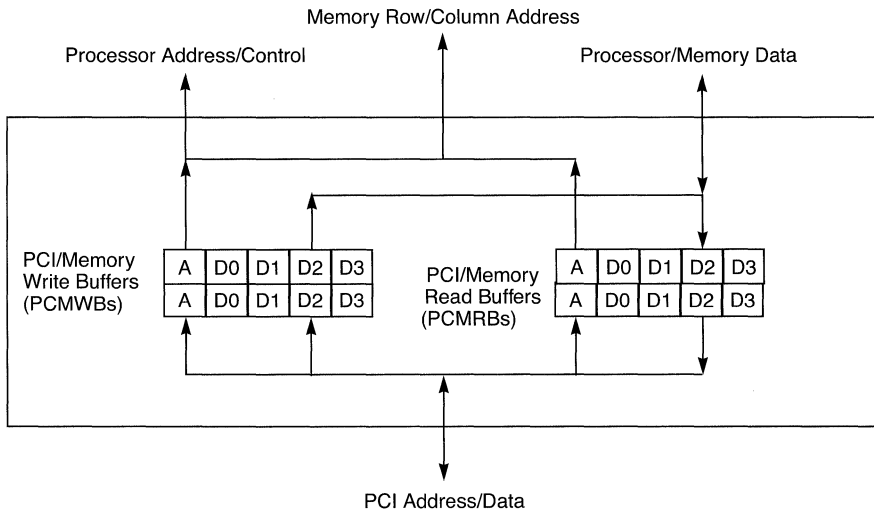


Figure 7-4. PCI/System Memory Buffers

Note

Many PCI accesses to system memory are snooped on the peripheral logic bus to ensure coherency between the PCI bus, system memory, and the L1 cache of the processor. All snoops for PCI accesses to system memory are performed strictly in-order.

Table 7-1 summarizes the snooping behavior of PCI accesses to system memory that hit in one of the internal buffers.

Table 7-1. Snooping Behavior Caused by a Hit in an Internal Buffer

PCI Transaction	Hit in Internal Buffer	Snoop Required
Read	Copy-back	Yes
Read (not locked)	PCMRB	No
Read (first access of a locked transfer)*	PCMRB	Yes
Read (not locked)	PCMWB	No
Read (first access of a locked transfer)*	PCMWB	Yes
Write	Copy-back	Yes
Write	PCMRB	Yes
Write	PCMWB	No

* Only reads can start an exclusive access (locked transfer). The first locked transfer must be snooped so that the cache line in the L1 is invalidated.

7.1.3.1 PCI to System Memory Read Buffering

The following subsections describe the PCMRB buffer and capability of the MPC8240 to perform speculative PCI reads from system memory.

7.1.3.2 PCI-to-System-Memory-Read Buffers (PCMRBs)

When a PCI device initiates a read from system memory, the address is snooped on the peripheral logic bus (provided snooping is enabled). If the memory interface is available, the memory access is started simultaneously with the snoop. If the snoop results in a hit in the L1 cache, the MPC8240 cancels the system memory access.

Depending on the outcome of the snoop, the requested data is latched into either one of the 32-byte PCI-to-system-memory-read buffers (PCMRBs), or into both the copy-back buffer and a PCMRB (as described in Section 7.1.1, “Processor Core/System Memory Buffers”) as follows:

- If the snoop hits in the L1, the copy-back data is written to both the copy-back buffer and to PCMRB. The data is forwarded to the PCI bus from the PCMRB, and to system memory from the copy-back buffer.

Internal Buffers

- If the snoop does not hit in the L1, a PCMRB is filled from system memory with the entire corresponding cache line, regardless of whether the starting address of the PCI-initiated transaction was at a cache line boundary. The data is forwarded to the PCI bus from the PCMRB as the PCMRB is loaded (the CCU doesn't wait for the PCMRB to be full).

The data is forwarded to the PCI bus as soon as it is received, not when the complete cache line has been written into a PCMRB. The addresses for subsequent PCI reads are compared to the existing address, so if the new access falls within the same cache line and the requested data is already latched in the buffer, the data can be forwarded to the PCI without requiring a snoop or another memory transaction.

If a PCI write to system memory hits in a PCMRB, the PCMRB is invalidated and the address is snooped on the peripheral logic bus. If the processor core accesses the address in the PCMRB, the PCMRB is invalidated.

7.1.3.3 Speculative PCI Reads from System Memory

To minimize the latency for large block transfers, the MPC8240 provides the ability to perform speculative PCI reads from system memory. When speculative reading is enabled (or a PCI read multiple transfer requests data word 2 of a cache line), the MPC8240 starts the snoop of the next sequential cache line address. Once the speculative snoop response is known and the MPC8240 has completed the current PCI read, the data at the speculative address is fetched from system memory and loaded into the other PCMRB in anticipation of the next PCI request.

Speculative PCI reads are enabled on a per access basis by using the PCI memory-read-multiple command. Speculative PCI reads can be enabled for all PCI memory read commands (memory-read, memory-read-multiple, and memory-read-line) by setting bit 2 in PICR1.

The MPC8240 starts the speculative read operation only under the following conditions:

- $PICR1[2] = 1$ or the current PCI read access is from a memory read-multiple command.
- No internal buffer flushes are pending.
- The address does not cross a 4 Kbyte boundary.
- The access is not a locked transaction.
- There are no outstanding configuration register accesses from the processor.
- The access is to system memory space. (The MPC8240 does not perform speculative reads from system ROM space.)

7.1.3.4 PCI-to-System-Memory-Write Buffers (PCMWBs)

For PCI write transactions to system memory, the MPC8240 employs two PCMWBs. The PCMWBs hold up to one cache line (32 bytes) each. Before PCI data is transferred to system memory, the address must be snooped on the peripheral logic bus (if snooping is

enabled). The buffers allow for the data to be latched while waiting for a snoop response. The write data can be accepted without inserting wait states on the PCI bus. Also, two buffers allow a PCI master to write to one buffer, while the other buffer is flushing its contents to system memory. Both PCMWBs are capable of gathering for writes to the same cache line.

If the snoop on the peripheral logic bus hits modified data in the L1 cache, the snoop copy-back data is merged with the data in the appropriate PCMWB, and the full cache line is sent to memory. For the PCI memory-write-and-invalidate command, a snoop hit in the L1 cache invalidates any modified cache line without requiring a copy-back.

Note that a PCI transaction that hits in either of the PCMWBs does not require a snoop on the peripheral logic bus. However, if a PCI write address hits in a PCI-read-from-system-memory buffer (PCMRB), the central control unit invalidates the PCMRB and snoops the address on the peripheral logic bus.

When the PCI write is complete and the snooping is resolved, the data is flushed to memory at the first available opportunity.

For a stream of single-beat writes, the data for the first transaction is latched in the first buffer and the central control unit initiates the snoop transaction on the peripheral logic bus. For subsequent single-beat writes, gathering is possible if the incoming write is to the same cache line as the previously latched data. Gathering to the first buffer can continue until the buffer is scheduled to be flushed, or until a write occurs to a different address. If there is valid data in both buffers, further gathering is not supported until one of the buffers has been flushed.

7.2 Internal Arbitration

The arbitration for the PCI bus is performed externally. All processor-to-PCI transactions are performed strictly in-order. Also, all snoops for PCI accesses to system memory are performed in order (if snooping is enabled). However, the MPC8240 performs arbitration internally for the shared processor/memory data bus. The arbitration for the processor/memory data bus employs the priority scheme shown in Table 7-2.

Table 7-2. Internal Arbitration Priorities

Priority	Operation
1	A high-priority copy-back buffer flush due to one of the following: A PCI access to system memory hits in the copy-back buffer. A processor burst write to system memory with ECC enabled hits a nonsnooped address in the PCMWB. A processor burst write to system memory with ECC enabled hits a nonsnooped address in the PCMRB.
2	A PCI read from system memory (with snoop complete)
3	A processor read from system memory

Table 7-2. Internal Arbitration Priorities (Continued)

Priority	Operation
4	A high priority PCMWB flush due to one of the following: A PCI read hits in the PCMWB The PCMWB is full and another PCI write to system memory starts A processor to system memory read hits in the PCMWB A processor to system memory single-beat write hits in the PCMWB
5	A medium priority copy-back buffer flush due to one of the following: A processor read hits in the copy-back buffer. A processor single-beat write hits in the copy-back buffer. The copy-back buffer is full and new data needs to be written to it.
6	Normal processor transfers including the following: A processor write to system memory A snoop copy-back due to a PCI write snoop A processor read from PCI A processor write to PCI A copy-back buffer fill
7	A PCI read from system memory (with snoop not complete)
8	A low-priority copy-back buffer flush
9	A low-priority PCMWB flush
10	A PCMRB prefetch from system memory due to a speculative PCI read operation.

Chapter 8

PCI Bus Interface

The MPC8240's PCI interface complies with the *PCI Local Bus Specification*, rev 2.1 and follows the guidelines in the *PCI System Design Guide*, revision 1.0 for host bridge architecture.

It is well beyond the scope of this manual to document the intricacies of the PCI bus. This chapter provides a rudimentary description of the PCI bus operations. The specific emphasis is directed at how the MPC8240 implements the PCI bus. It is strongly advised that anyone designing a system incorporating PCI devices should refer to the *PCI Local Bus Specification*, rev 2.1, and the *PCI System Design Guide*, revision 1.0, for a thorough description of the PCI local bus.

NOTE

Much of the available PCI literature refers to a 16-bit quantity as a word and a 32-bit quantity as a double word. Since 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.

8.1 PCI Interface Overview

The PCI interface connects the processor core and local memory to the PCI bus to which I/O components are connected. The PCI bus uses a 32-bit multiplexed, address/data bus, plus various control and error signals. The PCI interface supports address and data parity with error checking and reporting. Internal buffers are provided for operations between the PCI bus and the processor core or local memory. Processor read and write operations each have a 32-byte buffer, and memory operations have two 32-byte read buffers, and two 32-byte write buffers. See Chapter 7, "Central Control Unit," for more information.

The PCI interface of the MPC8240 functions both as a master (initiator) and a target device. Internally, the PCI interface of the MPC8240 is controlled by two state machines (one for master and one for target) running independently of each other. This allows the MPC8240 to handle two separate PCI transactions simultaneously. For example, if the MPC8240, as an initiator, is trying to run a burst-write to a PCI device, it may get disconnected before finishing the transaction. If another PCI device is granted the PCI bus and requests a burst-

read from local memory, the MPC8240, as a target, can accept the burst-read transfer. When the MPC8240 is granted mastership of the PCI bus, the burst-write transaction continues.

As an initiator, the MPC8240 supports read and write operations to the PCI memory space, the PCI I/O space, and the 256-byte PCI configuration space. As an initiator, the MPC8240 also supports generating PCI special-cycle and interrupt-acknowledge transactions. As a target, the MPC8240 supports read and write operations to local memory and, in agent mode, read and write operations to the PCI configuration space.

The MPC8240 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 agent mode is supported only for address map B. See Section 8.7, “PCI Host and Agent Modes,” for more information.

While in agent mode, all of the PCI configuration registers in the MPC8240 can be programmed from the PCI bus. However, the PICRs, MICRs, and other configuration registers are not accessible from the PCI bus and must be programmed by the processor core. See Section 8.7.2, “Accessing the MPC8240 Configuration Space in Agent Mode,” for more information.

The PCI interface provides bus arbitration for the MPC8240 and up to five other PCI bus masters. The arbitration algorithm is a programmable two-level round-robin priority selector. The on-chip PCI arbiter can operate in both host and agent modes or it can be disabled to allow for an external PCI arbiter.

The MPC8240 also provides an address translation mechanism to map inbound PCI to local memory accesses and outbound processor core to PCI accesses. Address translation is required when the MPC8240 is operating in agent mode. Address translation is not supported in host mode. See Section 8.7.4, “PCI Address Translation Support,” for more information.

The interface can be programmed for either little-endian or big-endian formatted data, and provides data swapping, byte enable swapping, and address translation in hardware. See Appendix B, “Bit and Byte Ordering,” for more information on the bi-endian features of the MPC8240.

8.1.1 The MPC8240 as a PCI Initiator

Upon detecting a processor-to-PCI transaction, the MPC8240 requests the use of the PCI bus. For processor-to-PCI bus write operations, the MPC8240 requests mastership of the PCI bus when the processor completes the write operation on the internal peripheral logic bus. For processor-to-PCI read operations, the MPC8240 requests mastership of the PCI bus when it decodes that the access is for PCI address space.

Once granted, the MPC8240 drives the 32-bit PCI address (AD[31–0]) and the bus command (C/BE[3–0]) signals. The master interface supports reads and writes of up to 32 bytes without inserting master-initiated wait states.

The master part of the interface can initiate master-abort cycles, recognizes target-abort, target-retry, and target-disconnect cycles, and supports various device selection timings. The master interface does not run fast back-to-back or exclusive accesses.

8.1.2 The MPC8240 as a PCI Target

As a target, upon detection of a PCI address phase the MPC8240 decodes the address and bus command to determine if the transaction is for local memory. If the transaction is destined for local memory, the target interface latches the address, decodes the PCI bus command, and forwards them to an internal control unit. On writes to local memory, data is forwarded along with the byte enables to the internal control unit. On reads, four bytes of data are provided to the PCI bus and the byte enables determine which byte lanes contain meaningful data.

The target interface of the MPC8240 can issue target-abort, target-retry, and target-disconnect cycles. The target interface supports fast back-to-back transactions and exclusive accesses using the PCI lock protocol. The target interface uses the fastest device selection timing.

The MPC8240 supports data streaming to and from local memory. This means that the MPC8240 can accept or provide data to or from local memory as long as the internal PCI-to-system-memory-write-buffers (PCMWBs) or PCI-to-system-memory-read buffers (PCMRBs) are not filled. For more information about the internal buffers of the MPC8240, see Chapter 7, “Central Control Unit.”

There are two 32-byte PCMWBs and while one is filled from the PCI master, the other is flushed to local memory. Some memory operations (such as refresh) can stall the flushing of the PCMWBs. In that case, the MPC8240 issues a target disconnect when there is no more space remaining in the PCMWBs.

Burst reads from local memory are accepted with wait states inserted depending upon the timing of local memory devices. The MPC8240 has two 32-byte PCMRBs and can provide continuous data to a PCI master by flushing one PCMRB to the PCI master while the other is being filled from local memory.

8.1.3 PCI Signal Output Hold Timing

In order to meet minimum output hold specifications relative to PCI_SYNC_IN for both 33 MHz and 66 MHz PCI systems, the MPC8240 has a programmable output hold delay for PCI signals. The initial value of the output hold delay is determined by the values on the \overline{MCP} and CKE power-on reset configuration signals. Further output hold delay values are available by programming the PCI_HOLD_DEL value of the PMCR2 configuration register. Refer to Section 5.3.2, “Power Management Configuration Register 2 (PMCR2),” and the *MPC8240 Hardware Specification* for more information on these values and signal timing.

8.2 PCI Bus Arbitration

PCI bus arbitration is access-based. Bus masters must arbitrate for each access performed on the bus. The PCI bus uses a central arbitration scheme where each master has its own unique request ($\overline{\text{REQ}}$) output and grant ($\overline{\text{GNT}}$) input signal. A simple request-grant handshake is used to gain access to the bus. Arbitration for the bus occurs during the previous access so that no PCI bus cycles are consumed due to arbitration (except when the bus is idle).

The MPC8240 provides bus arbitration logic for the MPC8240 and up to five other PCI bus masters. The on-chip PCI arbiter is independent of host or agent mode. The on-chip PCI arbiter can operate in both host and agent modes or it can be disabled to allow for an external PCI arbiter.

A configuration signal (MAA2) sampled at reset ($\overline{\text{HRST_CTRL}}$), determines if the on-chip PCI arbiter is enabled (low) or disabled (high). The on-chip PCI arbiter can also be enabled or disabled by programming bit 15 of the PCI arbitration control register. Note that the sense of bit 15 is the inverse of the configuration signal (that is, when bit 15 = 1 the arbiter is enabled, and when bit 15 = 0 the arbiter is disabled).

If the on-chip PCI arbiter is enabled, a request-grant pair of signals is provided for each external master ($\overline{\text{REQ}}[0-4]$ and $\overline{\text{GNT}}[0-4]$) and for the internal master state machine of the MPC8240. If the on-chip PCI arbiter is disabled, the MPC8240 uses the $\overline{\text{GNT0}}$ signal as an output to issue its request to the external arbiter, and uses the $\overline{\text{REQ0}}$ signal as an input to receive its grant from the external arbiter.

8.2.1 PCI Bus Arbiter Operation

The following subsections describe the operation of the on-chip PCI arbiter.

The on-chip PCI arbiter uses a programmable two-level, round-robin arbitration algorithm. Each of the five external masters, plus the MPC8240, can be programmed for two priority levels, high or low, using the appropriate bits in the PCI arbitration control register. Within each priority group (high or low), the PCI bus grant is asserted to the next requesting device in numerical order, with the MPC8240 positioned before device 0.

Conceptually, the lowest priority device is the master that is currently using the bus, and the highest priority device is the device that follows the current master in numerical order and group priority. This is considered to be a fair algorithm, since a single device cannot prevent other devices from having access to the bus; it automatically becomes the lowest priority device as soon as it begins to use the bus. If a master is not requesting the bus, then its transaction slot is given to the next requesting device within its priority group.

A grant is awarded to the highest priority requesting device as soon as the current master begins a transaction; however, the granted device must wait until the bus is relinquished by the current master before initiating a transaction.

The grant given to a particular device may be removed and awarded to another, higher priority device, whenever the higher priority device asserts its request. If the bus is idle when a device requests the bus, then for one clock cycle the arbiter withholds the grant. The arbiter re-evaluates the priorities of all requesting devices and grants the bus to the highest priority device in the following clock cycle. This allows a turnaround clock when a higher priority device is using address stepping or when the bus is parked.

The low-priority group collectively has one bus transaction request slot in the high-priority group. Mathematically, if there are N high-priority devices, and M low-priority devices, then each high-priority device is guaranteed to get at least 1 of $N+1$ bus transactions, and each low priority device is guaranteed to get at least 1 of $(N+1) \times M$ bus transactions, with one of the low-priority devices receiving the grant in 1 of $N+1$ bus transactions. If all devices are programmed to the same priority level, or if there is only one device in the low priority group, then the arbitration algorithm defaults to each device receiving an equal number of bus grants, in round-robin sequence.

Figure 8-1 shows an example of the arbitration algorithm. Assume that several masters are requesting use of the bus. If there are two masters in the high priority group and three in the low priority group, then each high-priority master is guaranteed at least 1 out of 3 transaction slots, and each low-priority master is guaranteed 1 out of 9 transaction slots.

In Figure 8-1, the grant sequence (with all devices except device 4 requesting the bus and device 3 being the current master) is 0, 2, MPC8240, 0, 2, 1, 0, 2, 3, ... and repeating. If device 2 is not requesting the bus, then the grant sequence is 0, MPC8240, 0, 1, 0, 3, ... and repeating. If device 2 requests the bus when device 0 is conducting a transaction and the MPC8240 has the next grant, then the MPC8240 will have its grant removed and device 2 will be awarded the grant since device 2 is of higher priority than the MPC8240 when device 0 has the bus.

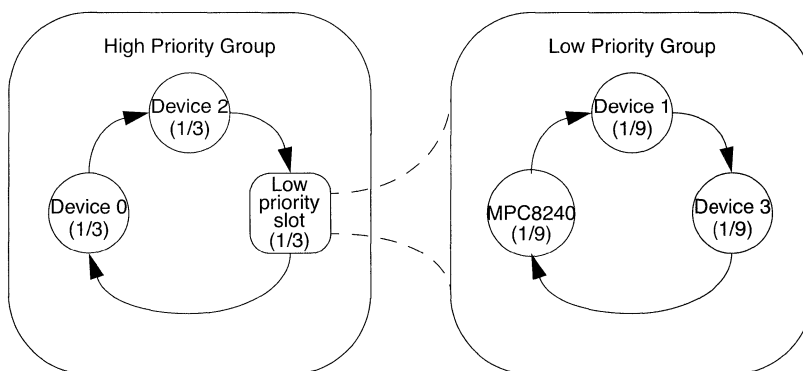


Figure 8-1. PCI Arbitration Example

8.2.2 PCI Bus Parking

When no device is using or requesting the bus, the PCI arbiter grants the bus to a selected device. This is known as parking the bus on the selected device. The selected device is required to drive the AD[31–0], $\overline{C}/\overline{BE}$ [0–3] and PAR signals to a stable value, preventing these signals from floating.

The parking mode control parameter (bits 14–13) in the PCI arbitration register determines which device the arbiter selects for parking the PCI bus. If parking mode control bits are 0b00 (or if the bus is not idle), then the bus is parked on the last master to use the bus. If the bus is idle, and the parking mode control bits are b10, then the bus is parked on the MPC8240; if the control bits are b01, then the bus is parked on device 0 (that is, the device connected to $\overline{GNT0}$).

8.2.3 Broken Master Lock-Out

The PCI bus arbiter on MPC8240 has a feature that allows it to lock out any masters that are broken or ill-behaved. The broken master feature is controlled by programming bit 12 of the PCI arbitration control register (0b0 = enabled, 0b1 = disabled).

When the broken master feature is enabled, a granted device that does not assert \overline{FRAME} within 16 PCI clock cycles after the bus is idle will have its grant removed and subsequent requests will be ignored until its \overline{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{REQ} signal. Disabling the broken master feature is not recommended.

8.2.4 Bus Lock Mode

Under certain conditions, it may be useful to have the bus remain granted to the owner of a resource lock, see Section 8.5, “Exclusive Access.” The bus lock mode parameter (bit 11) in the PCI arbitration control register alters the behavior of the PCI arbiter for locked bus operations.

If the bus lock mode bit is set, then for as long as \overline{LOCK} is asserted, the bus will be granted to the locking master. Note that this feature prevents any non-exclusive accesses while \overline{LOCK} is asserted which can reduce throughput in a system. Its use should be carefully considered.

8.2.5 Power-Saving Modes and the PCI Arbiter

In the sleep power-saving mode, the clock driving PCI_SYNC_IN can be disabled. If the clock is disabled, the arbitration logic is not able to perform its function. System programmers must park the bus with a device that can sustain the AD[31–0], $\overline{C}/\overline{BE}$ [0–3] and PAR signals prior to disabling the MPC8240's clock. If the bus is parked on the MPC8240 when its clocks are stopped, then the MPC8240 will sustain the AD[31–0], $\overline{C}/$

$\overline{\text{BE}}[0-3]$ and PAR signals in their prior states. In this situation, the only way for another agent to use the PCI bus is by waking the MPC8240. In nap and doze power-saving modes, the arbiter continues to operate allowing other PCI devices to run transactions.

8.3 PCI Bus Protocol

This section provides a general description of the PCI bus protocol. Specific PCI bus transactions are described in Section 8.4, “PCI Bus Transactions.” Refer to Figure 8-2, Figure 8-3, Figure 8-4, and Figure 8-5 for examples of the transfer-control mechanisms described in this section.

All signals are sampled on the rising edge of the PCI bus clock (PCI_SYNC_IN). Each signal has a setup and hold aperture with respect to the rising clock edge, in which transitions are not allowed. Outside this aperture, signal values or transitions have no significance.

8.3.1 Basic Transfer Control

The basic PCI bus transfer mechanism is a burst. A burst is composed of an address phase followed by one or more data phases. Fundamentally, all PCI data transfers are controlled by three signals— $\overline{\text{FRAME}}$ (frame), $\overline{\text{IRDY}}$ (initiator ready), and $\overline{\text{TRDY}}$ (target ready). An initiator asserts $\overline{\text{FRAME}}$ to indicate the beginning of a PCI bus transaction and negates $\overline{\text{FRAME}}$ to indicate the end of a PCI bus transaction. An initiator negates $\overline{\text{IRDY}}$ to force wait cycles. A target negates $\overline{\text{TRDY}}$ to force wait cycles.

The PCI bus is considered idle when both $\overline{\text{FRAME}}$ and $\overline{\text{IRDY}}$ are negated. The first clock cycle in which $\overline{\text{FRAME}}$ is asserted indicates the beginning of the address phase. The address and bus command code are transferred in that first cycle. The next cycle begins the first of one or more data phases. Data is transferred between initiator and target in each cycle that both $\overline{\text{IRDY}}$ and $\overline{\text{TRDY}}$ are asserted. Wait cycles may be inserted in a data phase by the initiator (by negating $\overline{\text{IRDY}}$) or by the target (by negating $\overline{\text{TRDY}}$).

Once an initiator has asserted $\overline{\text{IRDY}}$, it cannot change $\overline{\text{IRDY}}$ or $\overline{\text{FRAME}}$ until the current data phase completes regardless of the state of $\overline{\text{TRDY}}$. Once a target has asserted $\overline{\text{TRDY}}$ or $\overline{\text{STOP}}$, it cannot change $\overline{\text{DEVSEL}}$, $\overline{\text{TRDY}}$, or $\overline{\text{STOP}}$ until the current data phase completes. In simpler terms, once an initiator or target has committed to the data transfer, it cannot change its mind.

When the initiator intends to complete only one more data transfer (which could be immediately after the address phase), $\overline{\text{FRAME}}$ is negated and $\overline{\text{IRDY}}$ is asserted (or kept asserted) indicating the initiator is ready. After the target indicates the final data transfer (by asserting $\overline{\text{TRDY}}$), the PCI bus may return to the idle state (both $\overline{\text{FRAME}}$ and $\overline{\text{IRDY}}$ are negated) unless a fast back-to-back transaction is in progress. In the case of a fast back-to-back transaction, an address phase immediately follows the last data phase.

8.3.2 PCI Bus Commands

A PCI bus command is encoded in the $\overline{C}/\overline{BE}[3-0]$ signals during the address phase of a PCI transaction. The bus command indicates to the target the type of transaction the initiator is requesting. Table 8-1 describes the PCI bus commands as implemented by the MPC8240.

Table 8-1. PCI Bus Commands

$\overline{C}/\overline{BE}[3-0]$	PCI Bus Command	MPC8240 Supports as an Initiator	MPC8240 Supports as a Target	Definition
0000	Interrupt-acknowledge	Yes	No	The interrupt-acknowledge command is a read (implicitly addressing the system interrupt controller). Only one device on the PCI bus should respond to the interrupt-acknowledge command. Other devices ignore the interrupt-acknowledge command. See Section 8.4.6.1, "Interrupt Acknowledge Transactions," for more information.
0001	Special-cycle	Yes	No	The special-cycle command provides a mechanism to broadcast select messages to all devices on the PCI bus. See Section 8.4.6.2, "Special-Cycle Transactions," for more information.
0010	I/O-read	Yes	No	The I/O-read command accesses agents mapped into the PCI I/O space.
0011	I/O-write	Yes	No	The I/O-write command accesses agents mapped into the PCI I/O space.
0100	Reserved ¹	No	No	—
0101	Reserved ¹	No	No	—
0110	Memory-read	Yes	Yes	The memory-read command accesses either local memory, or agents mapped into PCI memory space, depending on the address. When a PCI master issues a memory-read command to local memory, the MPC8240 (the target) fetches data from the requested address to the end of the cache line (32 bytes) from local memory, even though all of the data may not be requested by (or sent to) the initiator.
0111	Memory-write	Yes	Yes	The memory-write command accesses either local memory, or agents mapped into PCI memory space, depending on the address.
1000	Reserved ¹	No	No	—
1001	Reserved ¹	No	No	—
1010	Configuration-read	Yes	Yes (if operating in agent mode)	The configuration-read command accesses the 256-byte configuration space of a PCI agent. A specific agent is selected when its IDSEL signal is asserted during the address phase. See Section 8.4.5, "Configuration Cycles," for more detail of PCI configuration cycles.

Table 8-1. PCI Bus Commands (Continued)

C/BE[3-0]	PCI Bus Command	MPC8240 Supports as an Initiator	MPC8240 Supports as a Target	Definition
1011	Configuration-write	Yes	Yes (if operating in agent mode)	The configuration-write command accesses the 256-byte configuration space of a PCI agent. A specific agent is selected when its IDSEL signal is asserted during the address phase. See Section 8.4.5.2, "Accessing the PCI Configuration Space," for more detail of PCI configuration accesses.
1100	Memory-read-multiple	Yes (for DMA cycles)	Yes	The memory-read-multiple command functions similar to the memory-read command, but it also causes a prefetch of the next cache line (32 bytes). Note that for PCI reads from local memory, prefetching for all reads may be forced by setting bit 2 (PCI speculative read enable) of PICR1. See Section 8.1.3.1.1, "Speculative PCI Reads from System Memory," for more information.
1101	Dual-address-cycle	No	No	The dual-address-cycle command is used to transfer a 64-bit address (in two 32-bit address cycles) to 64-bit addressable devices. The MPC8240 does not respond to this command.
1110	Memory-read-line	Yes	Yes	The memory-read-line command indicates that an initiator is requesting the transfer of an entire cache line (32 bytes).
1111	Memory-write-and-invalidate	Yes (for DMA cycles)	Yes	The memory-write-and-invalidate command indicates that an initiator is transferring an entire cache line (32 bytes), and, if this data is in any cacheable memory, that cache line needs to be invalidated.

¹Reserved command encodings are reserved for future use. The MPC8240 does not respond to these commands.

8.3.3 Addressing

PCI defines three physical address spaces—PCI memory space, PCI I/O space, and PCI configuration space. Access to the PCI memory and I/O space is straightforward, although one must take into account the MPC8240 address map (map A or map B) being used. The address maps are described in Chapter 4, "Address Maps." Access to the PCI configuration space is described in Section 8.4.5, "Configuration Cycles."

Address decoding on the PCI bus is performed by every device for every PCI transaction. Each agent is responsible for decoding its own address. PCI supports two types of address decoding—positive decoding and subtractive decoding. For positive decoding, each device is looking for accesses in the address range that the device has been assigned. For subtractive decoding, one device on the bus is looking for accesses that no other device has claimed. See Section 8.3.4, "Device Selection," for information about claiming transactions.

The information contained in the two low-order address bits (AD[1–0]) varies by the address space (memory, I/O, or configuration). Regardless of the encoding scheme, the two low-order address bits are always included in parity calculations.

8.3.3.1 Memory Space Addressing

For memory accesses, PCI defines two types of burst ordering controlled by the two low-order bits of the address—linear incrementing (AD[1–0] = 0b00) and cache wrap mode (AD[1–0] = 0b10). The other two AD[1–0] possibilities (0b01 and 0b11) are reserved. As a target, the MPC8240 executes a target disconnect after the first data phase completes if AD[1–0] = 0b01 or AD[1–0] = 0b11 during the address phase of a local memory access. As an initiator, the MPC8240 always encodes AD[1–0] = 0b00 for PCI memory space accesses.

For linear incrementing mode, the memory address is encoded/decoded using AD[31–2]. Thereafter, the address is incremented by 4 bytes after each data phase completes until the transaction is terminated or completed (a 4-byte data width per data phase is implied). Note that the two low-order bits of the address bus are still included in all parity calculations.

For cache wrap mode (AD[1–0] = 0b10) reads, the critical memory address is decoded using AD[31–2]. The address is incremented by 4 bytes after each data phase completes until the end of the cache line is reached. For cache-wrap reads, the address wraps to the beginning of the current cache line and continues incrementing until the entire cache line (32 bytes) is read. The MPC8240 does not support cache-wrap write operations and executes a target disconnect after the first data phase completes for writes with AD[1–0] = 0b10. Again, note that the two low-order bits of the address bus are still included in all parity calculations.

8.3.3.2 I/O Space Addressing

For PCI I/O accesses, all 32 address signals (AD[31–0]) are used to provide an address with granularity of a single byte. Once a target has claimed an I/O access, it must determine if it can complete the entire access as indicated by the byte enable signals. If all the selected bytes are not in the address range of the target, the entire access cannot complete. In this case, the target does not transfer any data, and terminates the transaction with a target-abort.

8.3.3.3 Configuration Space Addressing

PCI supports two types of configuration access, which use different formats for the AD[31–0] signals during the address phase. The two low-order bits of the address indicate the format used for the configuration address phase—type 0 (AD[1–0] = 0b00) or type 1 (AD[1–0] = 0b01). Both address formats identify a specific device and a specific configuration register for that device. See Section 8.4.5, “Configuration Cycles,” for descriptions of the two formats.

8.3.4 Device Selection

The $\overline{\text{DEVSEL}}$ signal is driven by the target of the current transaction. $\overline{\text{DEVSEL}}$ indicates

to the other devices on the PCI bus that the target has decoded the address and claimed the transaction. $\overline{\text{DEVSEL}}$ may be driven one, two, or three clock cycles (fast, medium, or slow device select timing) following the address phase. Device select timing is encoded into the device's PCI status register. If no agent asserts $\overline{\text{DEVSEL}}$ within three clock cycles of $\overline{\text{FRAME}}$, the agent responsible for subtractive decoding may claim the transaction by asserting $\overline{\text{DEVSEL}}$.

A target must assert $\overline{\text{DEVSEL}}$ (claim the transaction) before or coincident with any other target response (assert its $\overline{\text{TRDY}}$, $\overline{\text{STOP}}$, or data signals). In all cases except target-abort, once a target asserts $\overline{\text{DEVSEL}}$, it must not negate $\overline{\text{DEVSEL}}$ until $\overline{\text{FRAME}}$ is negated (with $\overline{\text{IRDY}}$ asserted) and the last data phase has completed. For normal termination, negation of $\overline{\text{DEVSEL}}$ coincides with the negation of $\overline{\text{TRDY}}$ or $\overline{\text{STOP}}$.

If the first access maps into a target's address range, that target asserts $\overline{\text{DEVSEL}}$ to claim the access. But, if the initiator attempts to continue the burst access across the resource boundary, then the target must issue a target disconnect.

The MPC8240 is hardwired for fast device select timing (PCI status register[10–9] = 0b00). Therefore, when the MPC8240 is the target of a transaction (local memory access or configuration register access in agent mode), it asserts $\overline{\text{DEVSEL}}$ one clock cycle following the address phase.

As an initiator, if the MPC8240 does not see the assertion of $\overline{\text{DEVSEL}}$ within four clock cycles after the address phase (that is, five clock cycles after it asserts $\overline{\text{FRAME}}$), it terminates the transaction with a master-abort.

8.3.5 Byte Alignment

The byte enable ($\overline{\text{C/BE}}[3-0]$, during a data phase) signals are used to determine which byte lanes carry meaningful data. The byte enable signals may enable different bytes for each of the data phases. The byte enables are valid on the edge of the clock that starts each data phase and stay valid for the entire data phase. Note that parity is calculated on all bytes regardless of the byte enables. See Section 8.6.1, "PCI Parity," for more information.

If the MPC8240, as a target, sees no byte enables asserted, it completes the current data phase with no permanent change. This implies that on a read transaction, the MPC8240 expects that the data is not changed, and on a write transaction, the data is not stored.

8.3.6 Bus Driving and Turnaround

A turnaround cycle is required, to avoid contention, on all signals that may be driven by more than one agent. The turnaround cycle occurs at different times for different signals. The $\overline{\text{IRDY}}$, $\overline{\text{TRDY}}$, $\overline{\text{DEVSEL}}$, and $\overline{\text{STOP}}$ signals use the address phase as their turnaround cycle. $\overline{\text{FRAME}}$, $\overline{\text{C/BE}}[3-0]$, and $\text{AD}[31-0]$ signals use the idle cycle between transactions (when both $\overline{\text{FRAME}}$ and $\overline{\text{IRDY}}$ are negated) as their turnaround cycle. The $\overline{\text{PERR}}$ signal has a turnaround cycle on the fourth clock after the last data phase.

The address/data signals, AD[31–0], are driven to a stable condition during every address/data phase. Even when the byte enables indicate that byte lanes carry meaningless data, the signals carry stable values. Parity is calculated on all bytes regardless of the byte enables. See Section 8.6.1, “PCI Parity,” for more information.

8.4 PCI Bus Transactions

This section provides descriptions of the PCI bus transactions. All bus transactions follow the protocol as described in Section 8.3, “PCI Bus Protocol.” Read and write transactions are similar for the memory and I/O spaces, so they are treated as a generic read transaction or a generic write transaction.

The timing diagrams show the relationship of significant signals involved in bus transactions. When a signal is drawn as a solid line, it is actively being driven by the current master or target. When a signal is drawn as a dashed line, no agent is actively driving it. High-impedance signals are indicated to have indeterminate values when the dashed line is between the two rails.

The terms ‘edge’ and ‘clock edge’ always refer to the rising edge of the clock. The terms ‘asserted’ and ‘negated’ always refer to the globally visible state of the signal on the clock edge, and not to signal transitions. ‘↻’ represents a turnaround cycle in the timing diagrams.

8.4.1 Read Transactions

This section refers to a PCI single-beat read transaction and a PCI burst read transaction.

The transaction starts with the address phase, occurring when an initiator asserts $\overline{\text{FRAME}}$. During the address phase, AD[31–0] contain a valid address and $\overline{\text{C/BE}}[3–0]$ contain a valid bus command.

The first data phase of a read transaction requires a turnaround cycle. This allows the transition from the initiator driving AD[31–0] as address signals to the target driving AD[31–0] as data signals. The turnaround cycle is enforced by the target using the $\overline{\text{TRDY}}$ signal. The target provides valid data, at the earliest one cycle after the turnaround cycle. The target must drive the address/data signals when $\overline{\text{DEVSEL}}$ is asserted.

During the data phase, the command/byte enable signals indicate which byte lanes are involved in the current data phase. A data phase may consist of a data transfer and wait cycles. The $\overline{\text{C/BE}}[3–0]$ signals remain actively driven for both reads and writes from the first clock of the data phase through the end of the transaction.

A data phase completes when data is transferred, which occurs when both $\overline{\text{IRDY}}$ and $\overline{\text{TRDY}}$ are asserted on the same clock edge. When either $\overline{\text{IRDY}}$ or $\overline{\text{TRDY}}$ is negated, a wait cycle is inserted and no data is transferred. The initiator indicates the last data phase by negating $\overline{\text{FRAME}}$ when $\overline{\text{IRDY}}$ is asserted. The transaction is considered complete when data is transferred in the last data phase.

Figure 8-2 illustrates a PCI single-beat read transaction. Figure 8-3 illustrates a PCI burst read transaction.

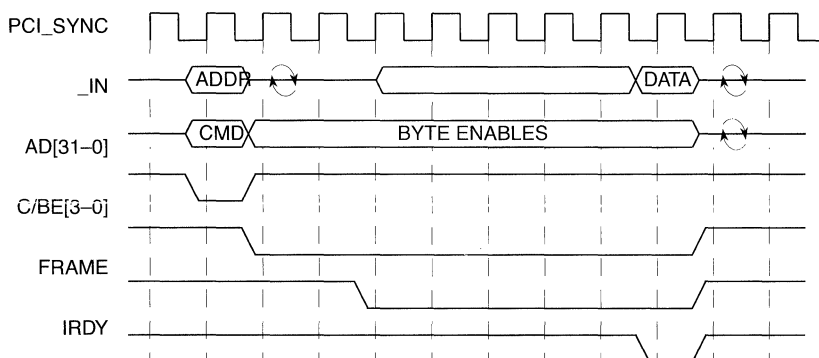


Figure 8-2. PCI Single-Beat Read Transaction

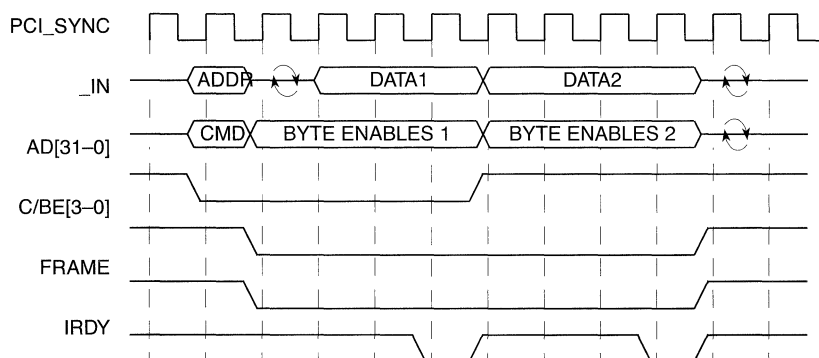


Figure 8-3. PCI Burst Read Transaction

8.4.2 Write Transactions

This section refers to a PCI single-beat write transaction, and a PCI burst write transaction. The transaction starts with the address phase, occurring when an initiator asserts $\overline{\text{FRAME}}$. A write transaction is similar to a read transaction except no turnaround cycle is needed following the address phase because the initiator provides both address and data. The data phases are the same for both read and write transactions. Although not shown in the figures, the initiator must drive the byte enable signals, even if the initiator is not ready to provide valid data ($\overline{\text{IRDY}}$ negated).

Figure 8-4 illustrates a PCI single-beat write transaction. Figure 8-5 illustrates a PCI burst write transaction.

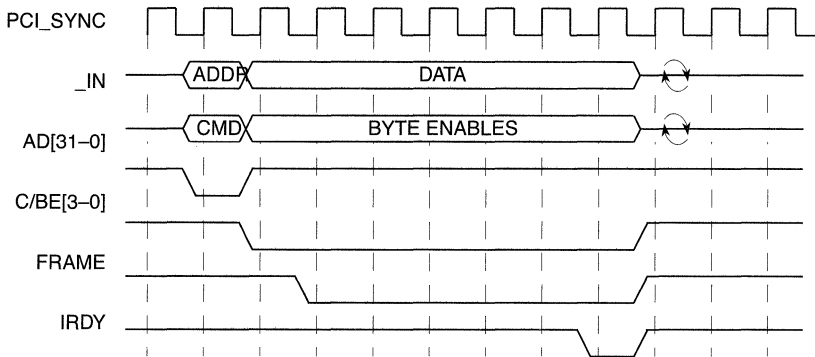


Figure 8-4. PCI Single-Beat Write Transaction

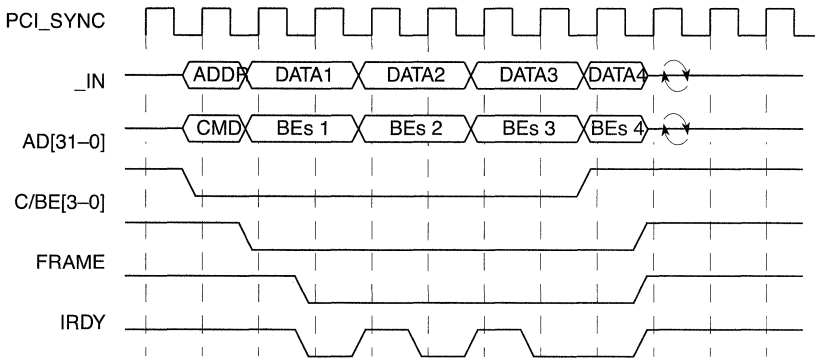


Figure 8-5. PCI Burst Write Transaction

8.4.3 Transaction Termination

Termination of a PCI transaction may be initiated by either the initiator or the target. The initiator is ultimately responsible for bringing all transactions to conclusion, regardless of the cause of the termination. All transactions are concluded when $\overline{\text{FRAME}}$ and $\overline{\text{IRDY}}$ are both negated, indicating the bus is idle.

8.4.3.1 Master-Initiated Termination

Normally, a master initiates termination by negating $\overline{\text{FRAME}}$ and asserting $\overline{\text{IRDY}}$. This indicates to the target that the final data phase is in progress. The final data transfer occurs when both $\overline{\text{TRDY}}$ and $\overline{\text{IRDY}}$ are asserted. The transaction is considered complete when data is transferred in the last data phase. After the final data phase, both $\overline{\text{FRAME}}$ and $\overline{\text{IRDY}}$ are negated (the bus becomes idle).

There are three types of master-initiated termination:

- **Completion**—Completion refers to termination when the initiator has concluded its intended transaction. This is the most common reason for termination.
- **Timeout**—Timeout refers to termination when the initiator loses its bus grant ($\overline{\text{GNT}}$ is negated), and its internal latency timer has expired. The intended transaction is not necessarily concluded.
- **Master-abort**—Master-abort is an abnormal case of master-initiated termination. If no device (including the subtractive decoding agent) asserts $\overline{\text{DEVSEL}}$ to claim a transaction, the initiator terminates the transaction with a master-abort. For a master-abort termination, the initiator negates $\overline{\text{FRAME}}$ and then negates $\overline{\text{IRDY}}$ on the next clock. If a transaction is terminated by master-abort (except for a special-cycle command), the received master-abort bit (bit 13) of the PCI status register is set.

As an initiator, if the MPC8240 does not detect the assertion of $\overline{\text{DEVSEL}}$ within four clock cycles following the address phase (five clock cycles after asserting $\overline{\text{FRAME}}$), it terminates the transaction with a master-abort. On reads that are master-aborted, the MPC8240 returns all 1s (0xFFFF). On writes that are master-aborted, the data is lost.

8.4.3.2 Target-Initiated Termination

By asserting the $\overline{\text{STOP}}$ signal, a target may request that the initiator terminate the current transaction. Once asserted, the target holds $\overline{\text{STOP}}$ asserted until the initiator negates $\overline{\text{FRAME}}$. Data may or may not be transferred during the request for termination. If $\overline{\text{TRDY}}$ and $\overline{\text{IRDY}}$ are asserted during the assertion of $\overline{\text{STOP}}$, data is transferred. However, if $\overline{\text{TRDY}}$ is negated when $\overline{\text{STOP}}$ is asserted, it indicates that the target will not transfer any more data; therefore, the initiator does not wait for a final data transfer as it would in a completion termination.

When a transaction is terminated by $\overline{\text{STOP}}$, the initiator must negate its $\overline{\text{REQ}}$ signal for a minimum of two PCI clock cycles, one being when the bus goes to the idle state ($\overline{\text{FRAME}}$ and $\overline{\text{IRDY}}$ negated). If the initiator intends to complete the transaction, it can reassert its $\overline{\text{REQ}}$ immediately following the two clock cycles. If the initiator does not intend to complete the transaction, it can assert $\overline{\text{REQ}}$ whenever it needs to use the PCI bus again.

There are three types of target-initiated termination:

- **Disconnect**—Disconnect refers to termination requested because the target is temporarily unable to continue bursting. Disconnect implies that some data has been transferred. the initiator may restart the transaction at a later time starting with the address of the next untransferred data. (That is, data transfer may resume where it left off.)
- **Retry**—Retry refers to termination requested because the target is currently in a state where it is unable to process the transaction. Retry implies that no data was transferred. The initiator may start the entire transaction over again at a later time. Note that the *PCI Local Bus Specification*, rev 2.1 requires that all retried transactions must be completed.

PCI Bus Transactions

- Target-abort—Target-abort is an abnormal case of target-initiated termination. Target-abort is used when a fatal error has occurred, or when a target will never be able to respond. Target-abort is indicated by asserting $\overline{\text{STOP}}$ and negating $\overline{\text{DEVSEL}}$. This indicates that the target requires termination of the transaction and does not want the transaction retried. If a transaction is terminated by target-abort, the received target-abort bit (bit 12) of the initiator's status register and the signaled target-abort bit (bit 11) of the target's status register is set. Note that any data transferred in a target-aborted transaction may be corrupt.

As a target, the MPC8240 terminates a transaction with a target disconnect due to the following:

- It is unable to respond within eight PCI clock cycles (not including the first data phase).
- A cache line (32 bytes) of data is transferred for a cache-wrap mode read transaction. (See the discussion of cache wrap mode in Section 8.3.3.1, "Memory Space Addressing," for more information.)
- A single beat of data is transferred for a cache-wrap mode write transaction. (See the discussion of cache wrap mode in Section 8.3.3.1, "Memory Space Addressing," for more information.)
- The last four bytes of a cache line are transferred in linear-incrementing address mode. (See the discussion of linear-incrementing mode in Section 8.3.3.1, "Memory Space Addressing," for more information.)
- If $\text{AD}[1-0] = 0b01$ or $\text{AD}[1-0] = 0b11$ during the address phase of a local memory access. (See Section 8.3.3.1, "Memory Space Addressing," for more information.)

As a target, the MPC8240 responds to a transaction with a retry due to the following:

- A processor copy-back operation is in progress.
- A PCI write to local memory was attempted when the internal PCI-to-local-memory-write buffers (PCMWBs) are full.
- A nonexclusive access was attempted to local memory while the MPC8240 is locked.
- A configuration write to a PCI device is underway and $\text{PICR2}[\text{NO_SERIAL_CFG}] = 0$.
- An access to one of the MPC8240 internal configuration registers is underway.
- The 32-clock latency timer has expired and the first data phase has not begun.

As a target, the MPC8240 responds with a target-abort if a PCI master attempts to write to the ROM/Flash ROM space in address map B. On PCI writes to local memory, if an address parity error or data parity error occurs, the MPC8240 aborts the transaction internally, but continues the transaction on the PCI bus.

Figure 8-6 shows several target-initiated terminations. The three disconnect terminations are unique in the data transferred at the end of the transaction. For Disconnect A, the

initiator is negating $\overline{\text{IRDY}}$ when the target asserts $\overline{\text{STOP}}$ and data is transferred only at the end of the current data phase. For Disconnect B, the target negates $\overline{\text{TRDY}}$ one clock after it asserts $\overline{\text{STOP}}$, indicating that the target can accept the current data, but no more data can be transferred. For the Disconnect-without-data, the target asserts $\overline{\text{STOP}}$ when $\overline{\text{TRDY}}$ is negated indicating that the target cannot accept any more data.

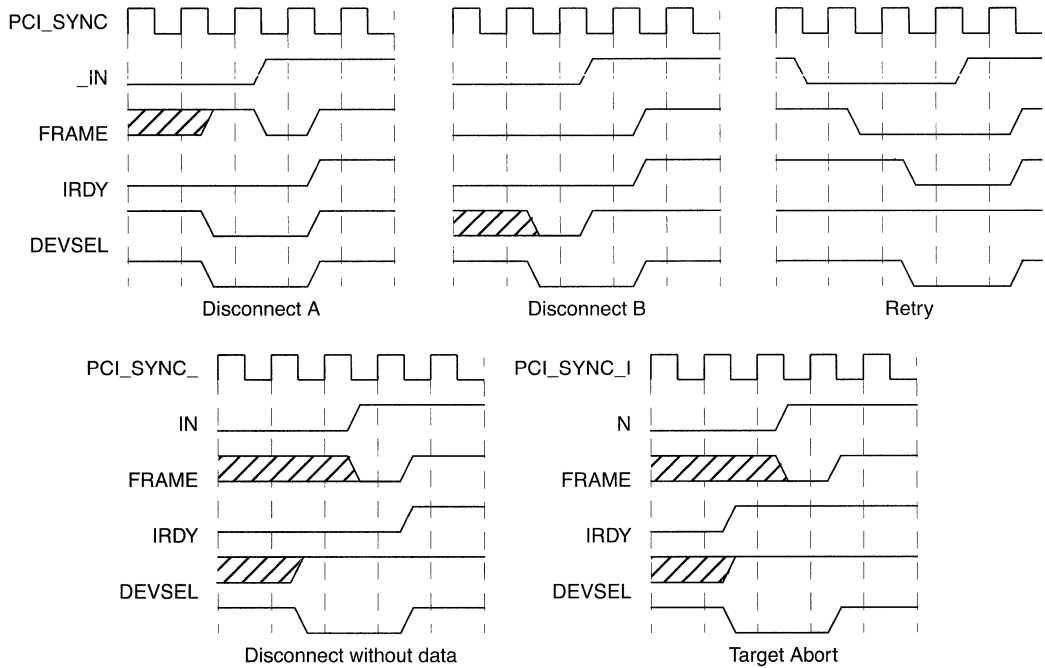


Figure 8-6. PCI Target-Initiated Terminations

8.4.4 Fast Back-to-Back Transactions

The PCI bus allows fast back-to-back transactions by the same master. During a fast back-to-back transaction, the initiator starts the next transaction immediately without an idle state. The last data phase completes when $\overline{\text{FRAME}}$ is negated, and $\overline{\text{IRDY}}$ and $\overline{\text{TRDY}}$ are asserted. The current master starts another transaction on the clock immediately following the last data transfer for the previous transaction.

Fast back-to-back transactions must avoid contention on the $\overline{\text{TRDY}}$, $\overline{\text{DEVSEL}}$, $\overline{\text{PERR}}$, and $\overline{\text{STOP}}$ signals. There are two types of fast back-to-back transactions — those that access the same target, and those that access multiple targets (sequentially). The first type places the burden of avoiding contention on the initiator; the second type places the burden of avoiding contention on all potential targets.

As an initiator, the MPC8240 does not perform any fast back-to-back transactions. As a target, the MPC8240 supports both types of fast back-to-back transactions.

During fast back-to-back transactions, the MPC8240 monitors the bus states to determine if it is the target of a transaction. If the previous transaction was not directed to the MPC8240 and the current transaction is directed at the MPC8240, the MPC8240 delays the assertion of DEVSEL (as well as TRDY, STOP, and PERR) for one clock cycle to allow the other target to stop driving the bus.

8.4.5 Configuration Cycles

This section describes configuring standard PCI devices using PCI configuration cycles. The PCI configuration space of any device is intended for configuration, initialization, and catastrophic error-handling functions only. Access to the PCI configuration space should be limited to initialization and error-handling software.

8.4.5.1 The PCI Configuration Space Header

The first 64 bytes of the 256-byte configuration space consists of a predefined header that every PCI device must support. The predefined header is shown in Figure 8-7. The rest of the 256-byte configuration space is device-specific.

The first 16 bytes of the predefined header are defined the same for all PCI devices; the remaining 48 bytes of the header may have differing layouts depending on the function of the device. Most PCI devices use the configuration header layout shown in Figure 8-7.

				Address Offset
Device ID		Vendor ID		00
Status		Command		04
Class Code			Revision ID	08
BIST	Header Type	Latency Timer	Cache Line Size	0C
Base Address Registers				10
				14
				18
				1C
				20
Reserved				24
Reserved				28
Reserved				2C
Expansion ROM Base Address				30
Reserved				34
Reserved				38
Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line	3C

Figure 8-7. Standard PCI Configuration Header

Table 8-2 summarizes the registers of the configuration header. Detailed descriptions of these registers are provided in the *PCI Local Bus Specification*, rev 2.1.

Table 8-2. PCI Configuration Space Header Summary

Address Offset	Register Name	Description
00	Vendor ID	Identifies the manufacturer of the device (assigned by the PCI SIG (special-interest group) to ensure uniqueness)
02	Device ID	Identifies the particular device (assigned by the vendor)
04	Command	Provides coarse control over a device's ability to generate and respond to PCI bus cycles
06	Status	Records status information for PCI bus-related events
08	Revision ID	Specifies a device-specific revision code (assigned by vendor)
09	Class code	Identifies the generic function of the device and (in some cases) a specific register-level programming interface
0C	Cache line size	Specifies the system cache line size in 32-bit units
0D	Latency timer	Specifies the value of the latency timer in PCI bus clock units for the device when acting as an initiator
0E	Header type	Bits 0–6 identify the layout of bytes 10–3F; bit 7 indicates a multifunction device. The most common header type (0x00) is shown in Figure 8-7 and in this table.
0F	BIST	Optional register for control and status of built-in self test (BIST)
10–27	Base address registers	Address mapping information for memory and I/O space
28	—	Reserved for future use
2C	—	Reserved for future use
30	Expansion ROM base address	Base address and size information for expansion ROM contained in an add-on board
34	—	Reserved for future use
38	—	Reserved for future use
3C	Interrupt line	Contains interrupt line routing information
3D	Interrupt pin	Indicates which interrupt pin the device (or function) uses
3E	Min_Gnt	Specifies the length of the device's burst period in 0.25 μ s units
3F	Max_Lat	Specifies how often the device needs to gain access to the bus in 0.25 μ s units

8.4.5.2 Accessing the PCI Configuration Space

When the MPC8240 is configured for host mode, the internal configuration registers are not accessible to external PCI agents. When the MPC8240 is configured for agent mode, external PCI agents have access to a subset of the internal configuration registers (see

Section 8.7.2, “Accessing the MPC8240 Configuration Space in Agent Mode.”). Configuring the internal registers of the MPC8240 in host mode and those registers that are not accessible in agent mode is described in Section 5.1, “Configuration Register Access.”

To support hierarchical bridges, two types of configuration accesses are supported. The first type of configuration access, type 0, is used to select a device on the local PCI bus. Type 0 configuration accesses are not propagated beyond the local PCI bus and must be claimed by a local device or terminated with a master-abort. The second type of configuration access, type 1, is used to pass a configuration request on to another PCI bus (through a PCI-to-PCI bridge). Type 1 accesses are ignored by all targets except PCI-to-PCI bridges.

To access the configuration space, a 32-bit value must be written to the CONFIG_ADDR register that specifies the target PCI bus, the target device on that bus, and the configuration register to be accessed within that device. A read or write to the CONFIG_DATA register causes the host bridge to translate the access into a PCI configuration cycle (provided the enable bit in CONFIG_ADDR is set and the device number is not 0b1_1111).

For the MPC8240, the CONFIG_ADDR register is located at different addresses depending on the memory address map in use. The address maps are described in Chapter 4, “Address Maps.” For address map A, the processor core can access the CONFIG_ADDR register through the MPC8240 at 0x8000_0CF8. For address map B, the processor can access the CONFIG_ADDR register at any location in the address range from 0xFEC0_0000 to 0xFEDF_FFFF. For simplicity, the address for CONFIG_ADDR is sometimes referred to as CF8, 0xnxxx_nCF8, or (in the PCI literature as) CF8h. Although systems implementing address map B can use any address in the range from 0xFEC0_0000 to 0xFEDF_FFFF for the CONFIG_ADDR register, the address 0xFEC0_0CF8 may be the most intuitive location.

The format of CONFIG_ADDR is shown in Figure 8-8.

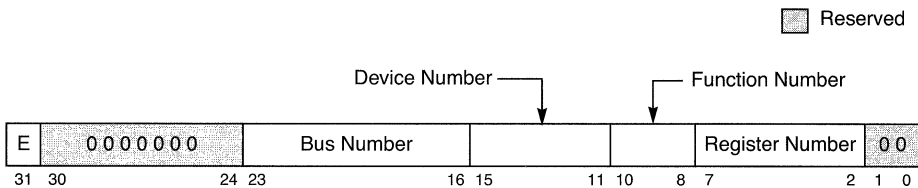


Figure 8-8. Layout of CONFIG_ADDR Register

Table 8-3 describes the fields within CONFIG_ADDR.

Table 8-3. CONFIG_ADDR Register Fields

Bits	Field Name	Description
31	E(enable)	The enable flag controls whether accesses to CONFIG_DATA are translated into PCI configuration cycles. 1 Enabled 0 Disabled

Table 8-3. CONFIG_ADDR Register Fields (Continued)

Bits	Field Name	Description
30–24	—	Reserved (must be 0b000_0000)
23–16	Bus number	This field is an encoded value used to select the target bus of the configuration access. For target devices on the PCI bus connected to the MPC8240, this field should be set to 0x00.
15–11	Device number	This field is used to select a specific device on the target bus.
10–8	Function number	This field is used to select a specific function in the requested device. Single-function devices should respond to function number 0b000.
7–2	Register number	This field is used to select the address offset in the configuration space of the target device.
1–0	—	Reserved (must be 0b00)

As with the CONFIG_ADDR register, the CONFIG_DATA register is located at different addresses depending on the memory address map in use. For address map A, the processor can access the CONFIG_DATA register through the MPC8240 at 0x8000_0CFC–0x8000_0CFF. For address map B, the processor can access the CONFIG_DATA register at any location in the address range from 0xFEE0_0000 to 0xFEEF_FFFF. For simplicity, the address for CONFIG_DATA is sometimes referred to as CFC, 0xnxxx_nCFC, or (in the PCI literature as) CFCh. Although systems implementing address map B can use any address in the range from 0xFEE0_0000 to 0xFEEF_FFFF for the CONFIG_DATA register, the address 0xFEE0_0CFC–0xFEE0_0CFF may be the most intuitive location.

Note that the CONFIG_DATA register may contain 1, 2, 3, or 4 bytes depending on the size of the register being accessed.

When the MPC8240 detects an access to the CONFIG_DATA register, it checks the enable flag and the device number in the CONFIG_ADDR register. If the enable bit is set, and the device number is not 0b1_1111, the MPC8240 performs a configuration cycle translation and runs a configuration-read or configuration-write transaction on the PCI bus. The device number 0b1_1111 is used for performing interrupt-acknowledge and special-cycle transactions. See Section 8.4.6, “Other Bus Transactions,” for more information. If the bus number corresponds to the local PCI bus (bus number = 0x00), the MPC8240 performs a type 0 configuration cycle translation. If the bus number indicates a remote PCI bus (that is, nonlocal), the MPC8240 performs a type 1 configuration cycle translation.

PCI Bus Transactions

Figure 8-9 shows the type 0 translation from the CONFIG_ADDR register to the AD[31-0] signals on the PCI bus during the address phase of the configuration cycle.

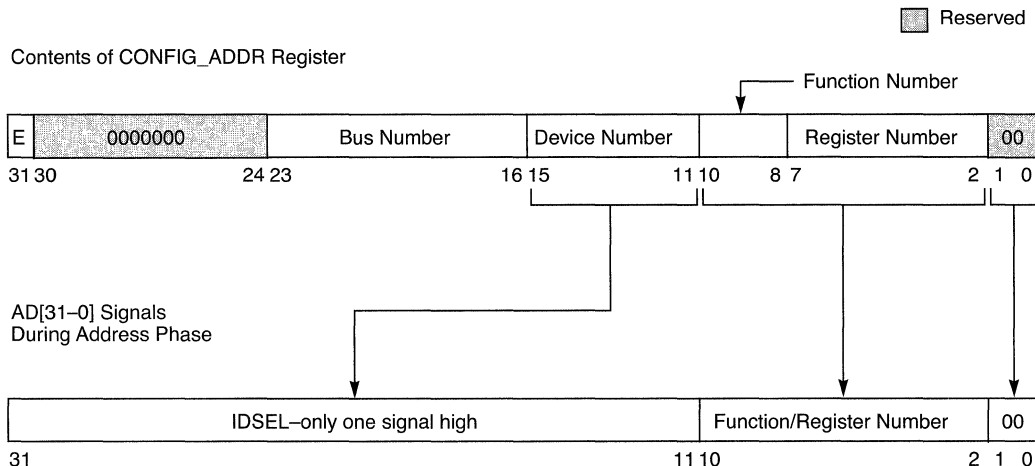


Figure 8-9. Type 0 Configuration Translation

For type 0 configuration cycles, the MPC8240 translates the device number field of the CONFIG_ADDR register into a unique IDSEL signal for up to 21 different devices. Each device connects its IDSEL input to one of the AD[31-11] signals. For type 0 configuration cycles, the MPC8240 translates the device number to IDSEL as shown in Table 8-4.

Table 8-4. Type 0 Configuration—Device Number to IDSEL Translation

Device Number		IDSEL
(Binary)	(Decimal)	
0b0_0000–0b0_1001	0–9	—
0b0_1010	10	AD31
0b0_1011	11	AD11
0b0_1100	12	AD12
0b0_1101	13	AD13
0b0_1110	14	AD14
0b0_1111	15	AD15
0b1_0000	16	AD16
0b1_0001	17	AD17
0b1_0010	18	AD18
0b1_0011	19	AD19

Table 8-4. Type 0 Configuration—Device Number to IDSEL Translation (Continued)

Device Number		IDSEL
(Binary)	(Decimal)	
0b1_0100	20	AD20
0b1_0101	21	AD21
0b1_0110	22	AD22
0b1_0111	23	AD23
0b1_1000	24	AD24
0b1_1001	25	AD25
0b1_1010	26	AD26
0b1_1011	27	AD27
0b1_1100	28	AD28
0b1_1101	29	AD29
0b1_1110	30	AD30
0b1_1111*	31	—

* A device number of all 1s indicates a PCI special-cycle or interrupt-acknowledge transaction

Note that the MPC8240 must not issue PCI configuration transactions to itself (that is, for PCI configuration transactions initiated by the MPC8240, its IDSEL input signal must not be asserted).

For type 0 translations, the function number and register number fields are copied without modification onto the AD[10–2] signals during the address phase. The AD[1–0] signals are driven to 0b00 during the address phase for type 0 configuration cycles.

For systems implementing address map A on the MPC8240, there is an alternate method for generating type 0 configuration cycles, called the direct access method. The direct access configuration method bypasses the CONFIG_ADDR translation step as shown in Figure 8-10.

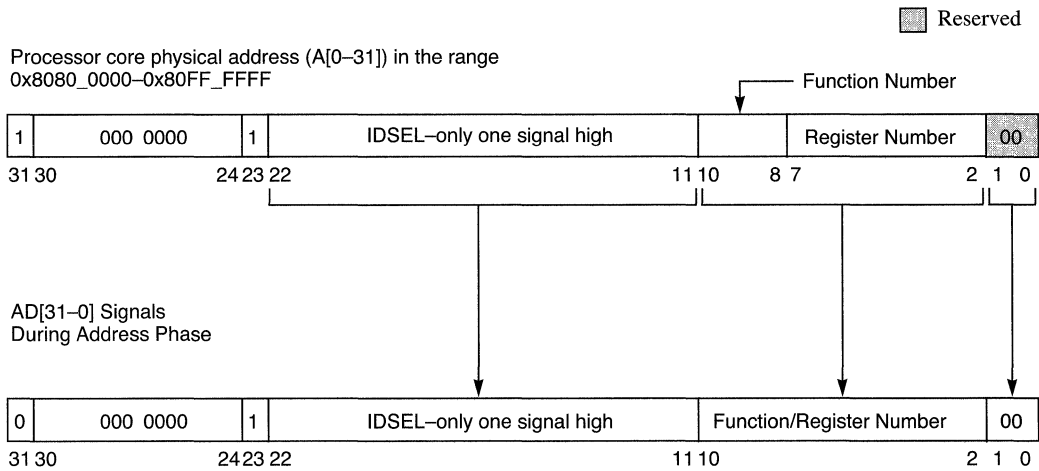


Figure 8-10. Direct-Access PCI Configuration Transaction

The MPC8240 decodes transactions from the processor core in address range from 0x8080_0000–0x80FF_FFFF, clears the most-significant address bit, and copies without modification the 30 low-order bits of the physical address onto the AD[31–0] signals during the address phase of a PCI configuration cycle. The two least-significant bits of the internal peripheral logic bus address, A[30–31], must be 0b00. Note that the direct-access method is limited to generating IDSEL on AD[11–22]. Also note that AD23 is always asserted for direct-access configuration cycles. Therefore, no PCI device should use AD23 for the IDSEL input on systems using address map A.

For type 1 translations, the MPC8240 copies without modification the 30 high-order bits of the CONFIG_ADDR register onto the AD[31–2] signals during the address phase. The MPC8240 automatically translates AD[1–0] into 0b01 during the address phase to indicate a type 1 configuration cycle.

8.4.6 Other Bus Transactions

There are two other PCI transactions that the MPC8240 supports—interrupt-acknowledge and special-cycles. As an initiator, the MPC8240 may initiate both interrupt-acknowledge and special-cycle transactions; however, as a target, the MPC8240 ignores interrupt-acknowledge and special-cycle transactions. Both transactions make use of the CONFIG_ADDR and CONFIG_DATA registers discussed in Section 8.4.5.2, “Accessing the PCI Configuration Space.”

8.4.6.1 Interrupt Acknowledge Transactions

The PCI bus supports an interrupt-acknowledge transaction. The interrupt-acknowledge command is a read operation implicitly addressed to the system interrupt controller. Note that the PCI interrupt-acknowledge command does not address the MPC8240’s EPIC

Processor Interrupt Acknowledge register and does not return the interrupt vector address from the EPIC unit. See Chapter 12, “Embedded Programmable Interrupt Controller (EPIC),” for more information about the EPIC unit.

When the MPC8240 detects a read to the CONFIG_DATA register, it checks the enable flag and the device number in the CONFIG_ADDR register. If the enable bit is set, the bus number corresponds to the local PCI bus (bus number = 0x00), the device number is all 1s (0b1_1111), the function number is all 1s (0b111), and the register number is zero (0b00_0000), then the MPC8240 performs an interrupt-acknowledge transaction. If the bus number indicates a nonlocal PCI bus, the MPC8240 performs a type 1 configuration cycle translation, similar to any other configuration cycle for which the bus number does not match.

The address phase contains no valid information other than the interrupt-acknowledge command ($\overline{C/BE}[3-0] = 0b0000$). There is no explicit address, however AD[31-0] are driven to a stable state and parity is generated. Only one device (the system interrupt controller) on the PCI bus should respond to the interrupt-acknowledge command by asserting \overline{DEVSEL} . All other devices on the bus should ignore the interrupt-acknowledge command. As stated previously, the MPC8240’s EPIC unit does not respond to PCI interrupt-acknowledge commands.

During the data phase, the responding device returns the interrupt vector on AD[31-0] when \overline{TRDY} is asserted. The size of the interrupt vector returned is indicated by the byte enable signals.

The MPC8240 also provides a direct method for generating PCI interrupt-acknowledge transactions. For address map A, processor reads to 0xBFFF_FFF0–0xBFFF_FFFF generate PCI interrupt acknowledge transactions. For address map B, processor reads to any location in the address range 0xFEFF_0000–0xFEFF_FFFF generate PCI interrupt-acknowledge transactions. Note that processor writes to these addresses cause processor transaction errors; see Section 13.3.1.1, “Processor Transaction Error,” for more information.

8.4.6.2 Special-Cycle Transactions

The special-cycle command provides a mechanism to broadcast select messages to all devices on the PCI bus. The special-cycle command contains no explicit destination address, but is broadcast to all PCI agents.

When the MPC8240 detects a write to the CONFIG_DATA register, it checks the enable flag and the device number in the CONFIG_ADDR register. If the enable bit is set, the bus number corresponds to the local PCI bus (bus number = 0x00), the device number is all 1s (0b1_1111), the function number is all 1s (0b111), and the register number is zero (0b00_0000), then the MPC8240 performs a special-cycle transaction on the local PCI bus. If the bus number indicates a nonlocal PCI bus, the MPC8240 performs a type 1 configuration cycle translation, similar to any other configuration cycle for which the bus number does not match.

Exclusive Access

The address phase contains no valid information other than the special-cycle command ($\overline{C/BE}[3-0] = 0b0001$). There is no explicit address, however $AD[31-0]$ are driven to a stable state and parity is generated. During the data phase, $AD[31-0]$ contain the special-cycle message and an optional data field. The special-cycle message is encoded on the 16 least-significant bits ($AD[15-0]$); the optional data field is encoded on the most-significant 16 lines ($AD[31-16]$). The special-cycle message encodings are assigned by the PCI SIG steering committee. The current list of defined encodings are provided in Table 8-5.

Table 8-5. Special-Cycle Message Encodings

AD[15-0]	Message
0x0000	SHUTDOWN
0x0001	HALT
0x0002	x86 architecture-specific
0x0003-0xFFFF	—

Note that the MPC8240 does not automatically issue a special-cycle message when it enters any of its power-saving modes. It is the responsibility of software to issue the appropriate special-cycle message, if needed.

Each receiving agent must determine whether the special-cycle message is applicable to itself. Assertion of \overline{DEVSEL} in response to a special-cycle command is not necessary. The initiator of the special-cycle transaction can insert wait states but since there is no specific target, the special-cycle message and optional data field are valid on the first clock \overline{IRDY} is asserted. All special-cycle transactions are terminated by master-abort; however, the master-abort bit in the initiator's status register is not set for special-cycle terminations.

8.5 Exclusive Access

PCI provides an exclusive access mechanism referred to as a resource lock. The mechanism locks only the selected PCI resource (typically memory) but allows other nonexclusive accesses to unlocked targets. In this section, the term 'locked operation' means an exclusive access to a locked target that may span several PCI transactions. A full description of exclusive access is contained in the *PCI Local Bus Specification*, rev 2.1

The \overline{LOCK} signal indicates that an exclusive access is underway. The assertion of \overline{GNT} does not guarantee control of the \overline{LOCK} signal. Control of \overline{LOCK} is obtained in conjunction with \overline{GNT} . When using resource lock, agents performing nonexclusive accesses are free to proceed even while another master retains ownership of \overline{LOCK} .

8.5.1 Starting an Exclusive Access

To initiate a locked operation, an initiator must receive \overline{GNT} when the \overline{LOCK} signal is not busy. the initiator is then said to own the \overline{LOCK} signal. To request a resource lock, the

initiator must hold $\overline{\text{LOCK}}$ negated during the address phase of a read command and assert $\overline{\text{LOCK}}$ in the clock cycle following the address phase. Note that the first transaction of a locked operation must be a read transaction.

The locked operation is not established on the PCI bus until the first data transfer ($\overline{\text{IRDY}}$ and $\overline{\text{TRDY}}$ asserted) completes. Once the lock is established, the initiator may retain ownership of the $\overline{\text{LOCK}}$ signal and the target may remain locked beyond the end of the current transaction. The initiator holds $\overline{\text{LOCK}}$ asserted until either the locked operation completes or until an error (master-abort or target-abort) causes an early termination. A target remains in the locked state until both $\overline{\text{FRAME}}$ and $\overline{\text{LOCK}}$ are negated. If the target retries the first transaction without a data phase completing, the initiator should not only terminate the transaction but should also release $\overline{\text{LOCK}}$.

8.5.2 Continuing an Exclusive Access

When the $\overline{\text{LOCK}}$ owner is granted access to the bus for another exclusive access to the previously-locked target, it negates the $\overline{\text{LOCK}}$ signal during the address phase to reestablish the lock. The locked target accepts the transaction and claims the transaction. The initiator asserts $\overline{\text{LOCK}}$ in the clock cycle following the address phase. If the initiator plans to continue the locked operation, it continues to assert $\overline{\text{LOCK}}$.

8.5.3 Completing an Exclusive Access

When an initiator is ready to complete an exclusive access, it should negate $\overline{\text{LOCK}}$ when $\overline{\text{IRDY}}$ is negated following the completion of the last data phase of the locked operation. This is to insure that the target is released prior to any other operation, and to insure that the resource is no longer blocked.

8.5.4 Attempting to Access a Locked Target

If $\overline{\text{LOCK}}$ is asserted during the address phase to a locked target, the locked target signals a retry, terminating the transaction without transferring any data. (The lock master always negates $\overline{\text{LOCK}}$ during the address phase of a transaction to a locked target.) Nonlocked targets ignore the $\overline{\text{LOCK}}$ signal when decoding the address. This allows other PCI agents to initiate and respond to transactions while maintaining exclusive access to the locked target.

8.5.5 Exclusive Access and the MPC8240

As an initiator, the MPC8240 does not generate locked operations. As a target, the MPC8240 responds to locked operations by guaranteeing complete access exclusion to local memory from the point-of-view of the PCI bus. From the point of view of the processor core, only the cache line (32 bytes) of the transaction is locked.

If an initiator on the PCI bus asserts $\overline{\text{LOCK}}$ for a read transaction to local memory, the MPC8240 completes the snoop transactions for any previous PCI-to-local-memory write operations and performs a snoop transaction for the locked read operation on the internal

PCI Error Functions

peripheral logic bus. Subsequent processor core accesses to local memory, when $\overline{\text{LOCK}}$ is asserted, are permitted with the exception that if the processor core attempts to access addresses within the locked cache line, the MPC8240 will retry the processor until the locked operation is completed. If a locked operation covers more than one cache line (32 bytes), only the most recently accessed cache line is locked from the processor. Since a snoop transaction is required to establish a lock, the MPC8240 does not honor the assertion of $\overline{\text{LOCK}}$ when PICR1[NO_SNOOP_EN] is set.

8.6 PCI Error Functions

PCI provides for parity and other system errors to be detected and reported. This section describes generation and detection of parity and error reporting for the PCI bus.

The PCI command register and error enabling registers 1 and 2 provide for selective enabling of specific PCI error detection. The PCI status register, error detection registers 1 and 2, the PCI bus error status register, and the 60x/PCI error address register provide PCI error reporting. These registers are described in Section 5.2, “PCI Interface Configuration Registers,” and Section 5.8, “Address Map B Options Register.”

8.6.1 PCI Parity

Generating parity is not optional; it must be performed by all PCI-compliant devices. All PCI transactions, regardless of type, calculate even parity; that is, the number of 1s on the AD[31–0], $\overline{\text{C/BE}}$ [3–0], and PAR signals all sum to an even number.

Parity provides a way to determine, on each transaction, if the initiator successfully addressed the target and transferred valid data. The $\overline{\text{C/BE}}$ [3–0] signals are included in the parity calculation to insure that the correct bus command is performed (during the address phase) and that correct data is transferred (during the data phase). The agent responsible for driving the bus must also drive even parity on the PAR signal one clock cycle after a valid address phase or valid data transfer, as shown in Figure 8-11.

During the address and data phases, parity covers all 32 address/data signals and the four command/byte enable signals regardless of whether all lines carry meaningful information. Byte lanes not actually transferring data must contain stable (albeit meaningless) data and are included in parity calculation. During configuration, special-cycle, or interrupt-acknowledge commands, some address lines are not defined but are driven to stable values and are included in parity calculation.

Agents that support parity checking must set the detected parity error bit in the PCI status register when a parity error is detected. Any additional response to a parity error is controlled by the parity error response bit in the PCI command register. If the parity error response bit is cleared, the agent ignores all parity errors.

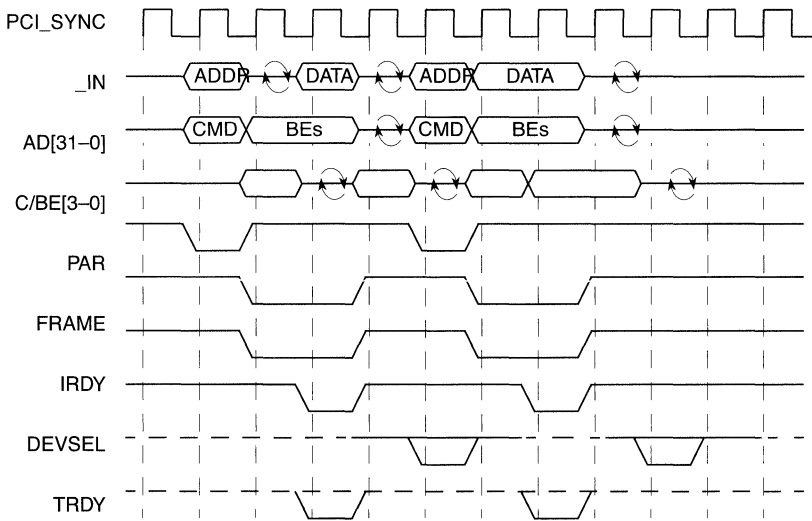


Figure 8-11. PCI Parity Operation

8.6.2 Error Reporting

PCI provides for the detection and signaling of both parity and other system errors. Two signals are used to report these errors— $\overline{\text{PERR}}$ and $\overline{\text{SERR}}$. The $\overline{\text{PERR}}$ signal is used exclusively to report data parity errors on all transactions except special-cycles. The $\overline{\text{SERR}}$ signal is used for other error signaling including address parity errors, data parity errors on special-cycle transactions, and may be used to signal other system errors. Refer to Section 13.3.3, “PCI Interface,” for a complete description of MPC8240 actions due to parity and other errors.

8.7 PCI Host and Agent Modes

The MPC8240 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’). The MAA1 configuration signal, sampled at reset, configures the MPC8240 for host or agent mode as described in Section 3.4, “Configuration Pins Sampled at Reset”. Note that agent mode is supported only for address map B. It is considered a configuration error when the MPC8240 is set to use address map A and agent mode. Also note that, in agent mode, the MPC8240 ignores all PCI memory accesses until inbound address translation is enabled. See Section 8.7.4.1, “Inbound PCI Address Translation,” and Section 4.3.1, “Inbound PCI Address Translation,” for more information about inbound address translation.

This section describes the different modes available in the MPC8240.

8.7.1 PCI Initialization Options

The assertion of the $\overline{\text{HRST_CPU}}$ and $\overline{\text{HRST_CTRL}}$ signals (must be asserted together) cause the processor to take a hard reset exception. The physical address of the handler is always 0xFFFF0_0100. Host and agent mode provide different options for initial reset exception vector fetching, and register configuration.

When the MPC8240 is configured for host mode, the system may initialize by fetching initial instructions from a ROM/Flash device. The $\overline{\text{RCS0}}$ configuration signal determines whether ROM/Flash is located in local memory space or in PCI memory space. See Section 3.4, “Configuration Pins Sampled at Reset,” for more information.

When the MPC8240 is configured for agent mode, it can also be configured to initialize from local memory space or remote PCI memory space.

Table 8-6 summarizes the initialization modes of the MPC8240. The initial settings of the PCI command register bus master and memory space bits (bits 2 and 1, respectively) are determined based on the reset configuration signals as shown in the table.

Table 8-6. Initialization Options for PCI Controller

Bus Master Mode (MAA1 at Reset)	ROM Location (RCS0 at Reset)	Initial Settings of PCI Command Register, and Boot Vector Fetch
Host (MAA1 high)	Local memory space (RCS0 high)	PCI command register [2,1] set to 10 Master enabled, target disabled. Boot vector fetch is sent to ROM located on the local memory interface.
Host (MAA1 high)	PCI memory space (RCS0 low)	PCI command register [2,1] set to 10 Master enabled, target disabled. Boot vector fetch is sent to PCI, and is issued on the bus unaltered.
Agent (MAA1 low)	Local memory space (RCS0 high)	PCI command register [2,1] set to 00 Master disabled, target disabled. Boot vector fetch is sent to ROM located on the local memory bus. Processor core configures local memory and has the option to set bit 10 of the PCI arbiter control register (RTY_PCI_CFG) to force PCI configuration cycles to be retried until local configuration is complete (see Section 8.7.3, “PCI Configuration Cycle Retry Capability in Agent Mode”). The MPC8240 can not issue transactions on the PCI bus until the master enable bit is set.
Agent (MAA1 low)	PCI memory space (RCS0 low)	PCI command register [2,1] set to 00 Master disabled, target disabled. Boot vector fetch is sent to PCI bus where it is not allowed to proceed until the host CPU enables bus mastership for the MPC8240 in the PCI control register. The processor core then proceeds sending the boot vector fetch to the PCI bus unaltered.

8.7.2 Accessing the MPC8240 Configuration Space in Agent Mode

When the MPC8240 is configured for agent mode, it responds to PCI configuration accesses from external hosts when the MPC8240's IDSEL input signal is asserted. This allows a remote host access to a subset of the MPC8240's internal configuration registers. Configuring the internal registers of the MPC8240 that are not accessible in agent mode is described in Section 5.1, "Configuration Register Access."

When accessing the MPC8240's configuration registers, the external host performs the translation shown in Figure 8-9. The external host uses the appropriate device number to assert the MPC8240's IDSEL input, and the desired function/register number from 0x00 to 0x47 as described in Section 5.1.5.2, "PCI-Accessible Configuration Registers."

Note that the MPC8240 must not issue PCI configuration transactions to itself (that is, for PCI configuration transactions initiated by the MPC8240, its IDSEL input signal must not be asserted).

8.7.3 PCI Configuration Cycle Retry Capability in Agent Mode

When the MPC8240 is configured for agent mode and is initializing from ROM located on the local memory bus, it may be necessary to defer a remote host from completing PCI configuration cycles until the local device can be tested and configured.

When the MPC8240 RTY_PCI_CFG bit (bit 10 in the PCI arbiter control register) is set, the MPC8240's PCI bus interface retries PCI configuration cycles. This mechanism allows the processor core to complete configuration of the local memory controller in advance of a system host controller. Once the MPC8240 has completed local configuration, it can clear the RTY_PCI_CFG bit, enabling the system host controller to complete configuration.

8.7.4 PCI Address Translation Support

The MPC8240 allows remapping PCI memory space transactions to local memory and processor core transactions to PCI memory space. Note that address translation is supported only for agent mode; it is not supported when the MPC8240 is operating in host mode. Also note that since agent mode is supported only for address map B, address translation is supported only for address map B. The following sections summarize the address translation support of the MPC8240. See Section 4.3, "Address Translation," for more detailed information about the MPC8240 address translation facility.

8.7.4.1 Inbound PCI Address Translation

Inbound transactions are PCI memory space accesses initiated by an external PCI master that are targeted toward the MPC8240. Using inbound address translation, the MPC8240 claims the PCI memory space transaction and translates it to a local memory access. When the MPC8240 is in agent mode, inbound address translation allows an external PCI master to access local memory through a window in the PCI memory space.

Note that in agent mode, the MPC8240 ignores all PCI accesses to local memory until inbound address translation is enabled. That is, in agent mode, the MPC8240 responds only to PCI configuration accesses and accesses to the embedded utilities memory block (EUMB) until inbound translation is enabled. See Section 4.3.1, “Inbound PCI Address Translation,” for a complete description of inbound PCI address translation.

8.7.4.2 Outbound PCI Address Translation

Outbound transactions are accesses initiated by the processor core that are targeted to PCI memory space. Using outbound address translation, the processor transaction is translated to an address in PCI memory space. When the MPC8240 is in agent mode, outbound address translation allows the MPC8240 to access (external) host memory in the lower 2 Gibbets of PCI memory space. See Section 4.3.2, “Outbound PCI Address Translation,” for a complete description of outbound PCI address translation.

8.7.4.3 Initialization Code Translation in Agent Mode

Since the processor always vectors to 0xFFFF0_0100 after a hard reset, it may be desirable in some systems to fetch from an alternate or translated address space. This allows a system designer to place the initialization code at some alternate system memory location such as system main memory or alternate system ROM space. When configured for agent mode, outbound PCI address translation is used to accomplish this task.

The MPC8240 has the flexibility of being programmable from a remote host controller. In this case, the outbound translation window is set to map the local ROM space to an alternate system address. The following procedure must be followed to take advantage of this functionality:

1. MPC8240 is configured for agent mode, with ROM located in PCI memory space.
2. System performs a hard reset.
3. The MPC8240 processor core fetches the hard reset exception vector, which is directed to the PCI bus. The transaction stalls and can not proceed until the PCI command register master enable bit is enabled.
4. The system host controller initializes and configures the MPC8240 as an agent.
5. The host must program PCSRBAR to locate the EUMB within PCI memory space.
6. The host must set bit 1 of the PCI command register to enable MPC8240 response to PCI memory accesses.
7. The host programs the outbound translation window to contain the ROM space and the outbound translation base address to point to the location in system (PCI memory) space where the initialization code resides.

8. The host then sets the PCI control register master enable bit in the MPC8240 to allow the local processor reset vector fetch (stalled in step 3) to initiate a read from the translated PCI location (as set up in step 7).
9. The MPC8240 then completes the pending reset exception fetch from the translated system address and configures the local memory registers (described in Section 5.9, “Memory Interface Configuration Registers”) and the inbound translation registers (ITWR and LMBAR) as described in Section 4.3.3, “Address Translation Registers.”

Chapter 9

DMA Controller

This chapter describes the DMA controller of the MPC8240. It includes the operation of the two DMA channels, the function of the DMA transfer types, the DMA descriptors' format, and the programming details for the DMA registers and their features.

9.1 DMA Overview

The MPC8240's DMA controller transfers blocks of data independent of the local processor or PCI hosts. Data movement occurs on the PCI and/or memory bus. The MPC8240 has two DMA channels, each with a 64-byte queue to facilitate the gathering and sending of data. Both the local processor and PCI masters can initiate a DMA transfer. Some of the features of the MPC8240 DMA unit are:

- Two DMA channels (0 and 1)
- Both channels accessible by processor core and remote PCI masters
- Misaligned transfer capability
- Scatter gather
- DMA chaining and direct DMA modes
- Interrupt on completed segment, chain, and error
- Four DMA transfer types:
 - Local memory to local memory
 - PCI memory to PCI memory
 - PCI memory to local memory
 - Local memory to PCI memory

Figure 9-1 provides a block diagram of the MPC8240 DMA unit.

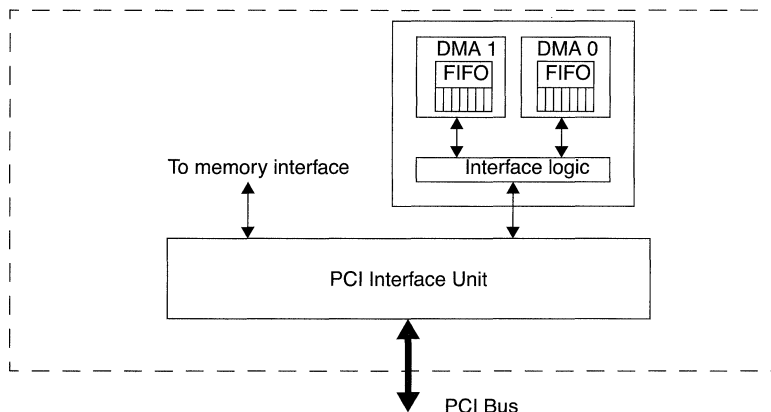


Figure 9-1. DMA Controller Block Diagram

9.2 DMA Register Summary

There are two complete sets of DMA registers on the MPC8240—one set for channel 0 and one set for channel 1. The two DMA channels on the MPC8240 are identical, except that the registers for channel 0 are located at offsets 0x100 and 0x0_1100, and the registers for channel 1 are located at offsets 0x200 and 0x0_1200. Throughout this chapter, the registers are described by a singular acronym (for example, DMR stands for the mode register for either channel 0 or channel 1). Table 9-1 summarizes the DMA registers on the MPC8240. All DMA registers are 32 bits wide.

Table 9-1. DMA Register Summary

PCI Memory Offset	Local Memory Offset	Register Name	Description
0x100	0x0_1100	DMA 0 Mode Register	Allows software to setup up different DMA modes and interrupt enables
0x104	0x0_1104	DMA 0 Status Register	Keeps track of DMA processes and errors
0x108	0x0_1108	DMA 0 Current Descriptor Address Register (CDAR)	Contains the location of the current descriptor to be loaded
0x110	0x0_1110	DMA 0 Source Address Register (SAR)	Contains the source address from which data will be read
0x118	0x0_1118	DMA 0 Destination Address Register (DAR)	Contains the destination address to which data will be written

Table 9-1. DMA Register Summary (Continued)

PCI Memory Offset	Local Memory Offset	Register Name	Description
0x120	0x0_1120	DMA 0 Byte Count Register (BCR)	Contains the number of bytes to transfer
0x124	0x0_1124	DMA 0 Next Descriptor Address Register	Contains the next descriptor address
0x200	0x0_1200	DMA 1 Mode Register	Allows software to setup up different DMA modes and interrupt enables.
0x204	0x0_1204	DMA 1 Status Register	Keeps track of DMA processes and errors
0x208	0x0_1208	DMA 1 Current Descriptor Address Register	Contains the location of the current descriptor to be loaded
0x210	0x0_1210	DMA 1 Source Address Register	Contains the source address from which data will be read
0x218	0x0_1218	DMA 1 Destination Address Register	Contains the destination address to which data will be written
0x220	0x0_1220	DMA 1 Byte Count Register	Contains the number of bytes to transfer
0x224	0x0_1224	DMA 1 Next Descriptor Address Register	Contains the next descriptor address

9.3 DMA Operation

The DMA controller operates in two modes—direct and chaining. In direct mode, the software is responsible for initializing the source, destination, and byte count registers (BCRs). In chaining mode, the software must first build descriptors segments in local or remote memory. Then the current descriptor address register (CDAR) is initialized to point to the first descriptor in memory. In both modes, setting the channel enable bit starts the DMA transfer.

The DMA controller supports unaligned transfers for both the source and destination addresses. It gathers data beginning at the source address, and aligns it accordingly before sending it to the destination address. The DMA controller assumes that the source and destination addresses are valid PCI or local memory addresses.

All local memory read operations are non-pipelined cache line reads (32 bytes). The DMA controller selects the valid data bytes within a cache line when storing in its queue. Writing to local memory depends on the destination address and the number of bytes transferred. The DMA controller attempts to write cache lines (non-pipelined) if the destination address is aligned on a 32-byte boundary. Otherwise, partial cache line writes are performed.

PCI memory read operations depend on the PRC bits in the DMA mode register (DMR), the source address, and the number of bytes transferred. The DMA controller attempts to read a cache line whenever possible. Writing to PCI memory depends on the destination address and the number of bytes transferred. PCI Write-and-Invalidate operations are

performed only if a full cache line is being transferred, the PCI command status register (PCSR) bit 4 (memory write and invalidate) is set, and the PCI cache line size register is set to 0x08. Otherwise, write operations are performed.

The internal DMA protocols operate on a cache line basis, so the MPC8240 always attempts to perform transfers that are the size of a cache line. The only possible exceptions are the first or last transfer. To further enhance performance, the protocols also allow for multiple-cache-line-streaming operation in which more than one cache line can be transferred at one time. Therefore, maximum performance is achieved when the initial address of a DMA transfer is aligned to a cache-line boundary.

9.3.1 DMA Direct Mode

In direct mode, the DMA controller does not read descriptors from memory but instead uses the current parameters in the DMA registers to start a DMA process. The DMA transfer is finished after all bytes specified in the BCR have been transferred, or an error condition has occurred. The initialization steps for a DMA transfer in direct mode are shown below:

1. Poll the CB in the DMA status register (DSR) to make sure the DMA channel is idle.
2. Initialize the source, destination and BCR.
3. Initialize the CTT bit in the CDAR to indicate the type of transfer.
4. Initialize the CTM bit in the DMR to indicate direct mode. Other control parameters in the DMR can also be initialized here, if necessary.
5. Clear and then set the CS bit in the DMR to start the DMA transfer.

9.3.2 DMA 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 the 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 memory, or an error condition has occurred.

9.3.2.1 Basic Chaining Mode Initialization

The initialization steps for a DMA transfer in chaining mode are shown below:

1. Build descriptor segments in memory. Refer to the Section 9.5, “DMA Descriptors,” for more information.
2. Poll the CB bit in the DMA status register (DSR) to make sure the DMA channel is idle.
3. Initialize the CDAR to point to the first descriptor in memory.

4. Initialize the CTM bit in the DMR to indicate chaining mode. Other control parameters in the DMR can also be initialized here if necessary.
5. Clear and then set the CS bit in the DMR to start the DMA transfer.

9.3.2.2 Periodic DMA Feature

Periodic DMA is a feature that allows a DMA process to be repeated over and over again with the same parameters while in chaining mode. This feature has potential use in applications that require periodic movement of data. The MPC8240 uses two of the timers in the EPIC unit to automatically signal the DMA channels to start a DMA process, without the use of the processor interrupt. In this mode, timer2 automatically signals DMA channel 0 and timer3 automatically signals DMA channel 1. The following sequence describes the steps to set up the periodic DMA feature:

1. Set up timer2 (and/or timer3) with the mask bit set (when the timer counts down, no interrupt is generated to the processor). Program the timer to operate at the appropriate rate and clear the CI bit in the corresponding GTBCR. Choose a rate for the timer longer than the time required to complete the DMA chain; otherwise, unpredictable operation occurs.
2. Program the DMA channel to operate in chaining mode (described in Section 9.3.2.1, “Basic Chaining Mode Initialization”).
3. Enable periodic DMA by setting the PDE bit in the DMR.

When the timer first expires, the DMA hardware begins the data movement. In this mode, the current descriptor address is automatically saved for later use. When the timer expires the second time, the DMA reloads the saved current descriptor address into the CDAR and restarts. This process continues until an error condition occurs, or the timer is stopped.

9.3.3 DMA Coherency

Each DMA channel contains a 64-byte transfer queue. No address snooping occurs in these queues. It is therefore possible that certain data could be posted in these queues and not be visible to the rest of the system, while a DMA transfer is in progress. Therefore, software must ensure coherency of the region being transferred during the DMA process.

Snooping of the processor data cache is selectable during DMA transactions. A snoop bit is provided in the CDAR and the next descriptor address register (NDAR) which allows software to control when the cache is snooped. These bits are described in Section 9.6.3, “Current Descriptor Address Registers (CDARs)” and Section 9.6.7, “Next Descriptor Address Registers (NDARs)” respectively.

To improve performance, transactions from memory are prefetched. If a DMA descriptor chain is marked as no-snoop, any transaction to the 4-Kbyte region within which the transfer is occurring is not guaranteed to be snooped. If another device were to access another region of the 4-Kbyte page, it is not guaranteed that a snoop cycle would be issued to the local MPC8240 processor core.

9.4 DMA Transfer Types

The DMA controller supports four types of transfer—PCI to PCI; PCI to memory; memory to PCI; and memory to memory. All data is temporarily stored in a 64-byte DMA queue before transmission.

9.4.1 PCI to PCI

For PCI memory to PCI memory transfers, the DMA controller begins by reading data from PCI memory space and storing it in the DMA queue. When there is enough data in the queue, the DMA controller begins writing data to PCI memory space beginning at the destination address. The process is repeated until there is no more data to transfer, or an error condition has occurred on the PCI bus.

9.4.2 PCI to Local Memory

For PCI memory to local memory transfers, the DMA controller initiates reads on the PCI bus and stores the data in the DMA queue. Once sufficient data is stored in the queue, a local memory write is initiated. The DMA controller stops the transferring process either when there is an error condition on the PCI bus or local memory interface, or when no data is left to transfer. Reading from PCI memory and writing to local memory can occur concurrently.

9.4.3 Local Memory to PCI

For local memory to PCI memory transfers the DMA controller initially fetches data from local memory into the DMA queue. As soon as the first data arrives into the queue, the DMA engine initiates write transactions to PCI memory. The DMA controller stops the transferring process either when there is an error on the PCI bus or local memory interface, or when no data is left to transfer. Reading from local memory and writing to PCI memory can occur concurrently.

9.4.4 Local Memory to Local Memory

For local memory to local memory transfers, the DMA controller begins reading data from local memory and stores it in the DMA queue. When the queue is full, the DMA controller begins writing data to local memory space, beginning at the destination address. The process is repeated until there is no more data to transfer or an error condition occurs while accessing memory.

9.4.5 Address Map Interactions

Because of the flexibility of the MPC8240's DMA controller, certain interactions can occur related to the specific address map and mode in which MPC8240 is operating.

For either host or agent mode, if the outbound translation window is programmed over MPC8240's I/O space (0xFE0x_xxxx - 0xFEBx_xxxx), configuration space (0xFECx_xxxx - 0xFEDx_xxxx), or interrupt acknowledge space (0xFEEx_xxxx), then a DMA transaction to these address spaces results in a translated outbound address. Note that this differs from a processor-generated transaction. In the case of a processor-generated transaction to these spaces, these addresses' ranges appear as holes in the outbound translation window.

9.4.5.1 Host Mode Interactions

The following subsections describe five cases of interactions with the host mode address maps.

9.4.5.1.1 PCI Master Abort when PCI Bus Specified for Lower 2-Gbyte Space

If the MPC8240 is in host mode and a transferred address falls within the lower 2-Gbyte space (0x0000_0000 to 0x7FFF_FFFF) on the PCI bus (specified by CDAR[CTT]), then the MPC8240 issues the transaction to the PCI bus with that address. However, this address space is reserved for the host controller; therefore, no PCI slave responds to the transaction; the transaction terminates with a PCI master abort.

9.4.5.1.2 Address Alias to Lower 2-Gbyte Space

If the MPC8240 is in host mode, the CDAR[CTT] indicates that the transferred address is for local memory, and the address falls between 0x8000_0000 and 0xFEFF_FFFF, then the transaction is issued to local memory and the address is aliased to the lower 2-Gbyte space.

9.4.5.1.3 Attempted Writes to Local ROM/Port X Space

If the MPC8240 is in host mode, CDAR[CCT] indicates that the transferred address is for local memory, and the address falls between 0xFF00_0000 and 0xFFFF_FFFF, only read operations are allowed. Attempts to program the DMA controller to write to this space (considered the local ROM/Port X space) under these conditions results in the assertion of the internal *mcp* signal, and a machine check exception (if enabled). See Chapter 13, "Error Handling and Exceptions," for more information about the internal *mcp* signal.

9.4.5.1.4 Attempted Accesses to ROM on the PCI Bus—Host Mode

If the MPC8240 is in host mode, CDAR[CTT] indicates that the transferred address is for local ROM space, and the MPC8240 is configured for ROM on the PCI bus, the transaction is performed to the local ROM interface. Unknown data will be returned. (This is considered a programming error.)

9.4.5.1.5 Attempted Accesses to ROM on the Memory Bus

If the MPC8240 is in host mode, the CDAR[CTT] indicates that the transferred address is for PCI ROM space, and the MPC8240 is configured for ROM on the local memory interface, then the transaction is issued to the PCI bus. The transaction will result in either a master abort or an access to a configured device in the ROM address space on the PCI bus.

9.4.5.2 Agent Mode Interactions

The following subsections describe three cases of interactions with the agent mode address maps.

9.4.5.2.1 Agent Mode DMA Transfers for PCI

When CDAR[CTT] indicates that the transferred address is for PCI, any address can be issued within the 32-bit address space. If the software running on an MPC8240 configured as an agent is aware of the system address map, it can perform DMA transfers with the untranslated system address.

Alternatively, the MPC8240 agent DMA driver does not need to be aware of the system memory map, and it can rely on address translation to be performed by the ATU. If the address falls within the outbound translation window, then the address is translated through the ATU. In this case, transactions should be steered through the outbound translation window.

9.4.5.2.2 Attempted Accesses to Local ROM when ROM is on PCI

If the CDAR[CTT] indicates that the transferred address is for local ROM space and the ROM is located on PCI then the transaction is issued to local memory and results in unknown data returned.

9.4.5.2.3 Attempted Access to ROM on the PCI Bus—Agent Mode

If the CDAR[CTT] indicates that the transferred address is for ROM on the PCI bus and the ROM is located locally, then the transaction is issued to the PCI bus and the transaction results in a master abort or a completed transaction, depending on whether a device is configured on the PCI bus in that address space.

9.5 DMA Descriptors

DMA descriptors are constructed in either local or PCI memory and linked together by the next descriptor field. The descriptor contains information for the DMA controller to transfer data. Software must ensure that each segment descriptor is aligned on an 8-word boundary. The last descriptor in memory must have the EOTD bit set in the next descriptor field, indicating that this is the last descriptor in memory.

Software initializes the CDAR to point to the first descriptor in memory. The DMA controller traverses through the descriptor chain until the last descriptor is read. For each descriptor in the chain, the DMA controller starts a new DMA transfer with the control parameters specified by the descriptor.

Table 9-2 summarizes the fields of DMA descriptors.

Table 9-2. DMA Descriptor Summary

Descriptor Field	Description
Source Address	Contains the source address of the DMA transfer. As the DMA controller reads the descriptor from memory, this field is loaded into the SAR. The bit definition is the same as that described for the SAR in Table 9-5.
Destination Address	Contains the destination address of the DMA transfer. As the DMA controller reads the descriptor from memory, this field is loaded into the DAR. The bit definition is the same as that described for the DAR in Table 9-5.
Next Descriptor Address	Points to the next descriptor in memory. As the DMA controller reads the descriptor from memory, this field is loaded into the NDAR. The bit definition is the same as that of the NDAR described in Table 9-5. If the current descriptor is the last descriptor in memory, then the EOTD bit in this descriptor field should be set.
Byte Count	Contains the number of bytes to transfer. As the DMA controller reads the descriptor from memory, this field is loaded into the BCR. The bit definition is the same as that of the BCR described in Table 9-8.

DMA Descriptors

Figure 9-2 shows how the DMA descriptors in memory are chained together.

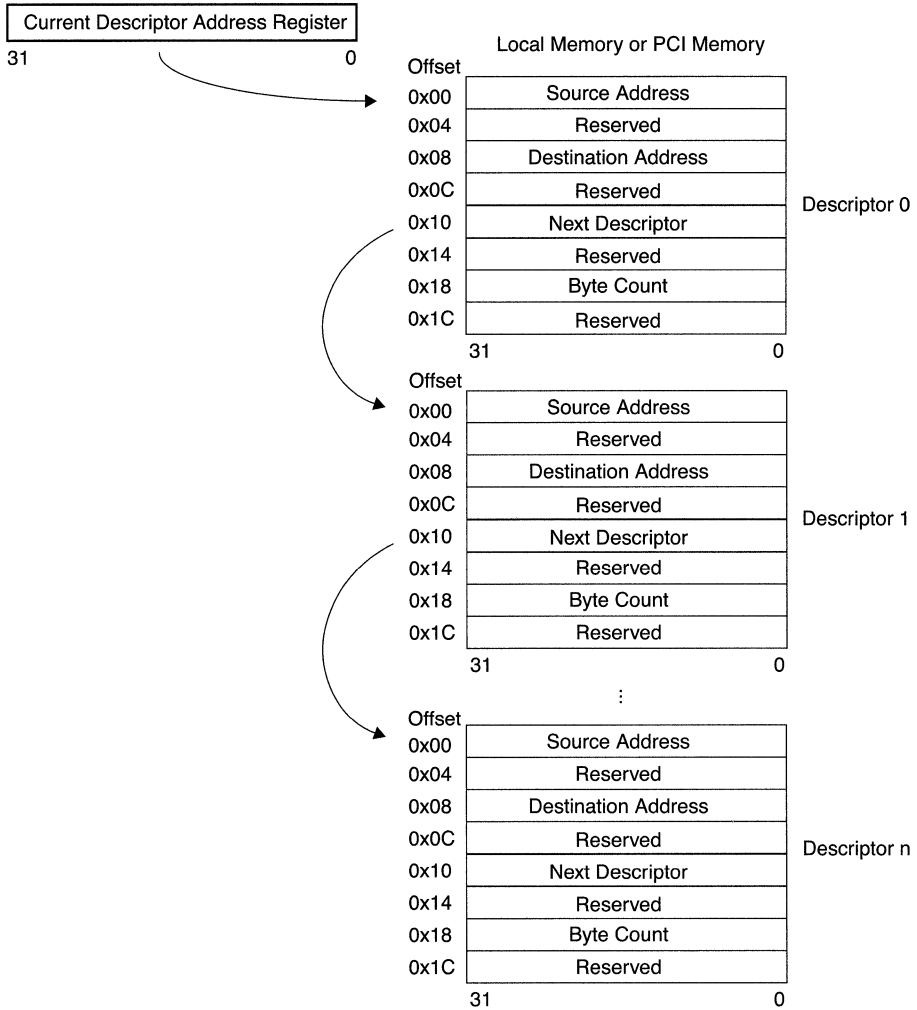


Figure 9-2. Chaining of DMA Descriptors in Memory

9.5.1 Descriptors in Big-Endian Mode

In big-endian byte ordering mode, the descriptors in local memory should be programmed with data appearing in ascending significant byte order.

For example, a big-endian mode descriptor's data structure is shown below:

```
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 = 0x12345678 <MSB..LSB>
        Byte Count = 0x76543210 <MSB..LSB>
```

Note that the descriptor struct must be aligned on an 8-word boundary.

9.5.2 Descriptors in Little-Endian Mode

In little-endian byte ordering mode, the descriptor in local memory should be programmed with data appearing in descending significant byte order.

For example, a little-endian mode descriptor's data structure is shown below:

```
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>
```

Note that the descriptor struct must be aligned on an 8-word boundary.

9.6 DMA Register Descriptions

The following sections describe the DMA controller registers and their bit settings in detail.

9.6.1 DMA Mode Registers (DMRs)

The DMRs allow software to start the DMA transfer and to control various DMA transfer characteristics. Figure 9-3 shows the bits in the DMRs.

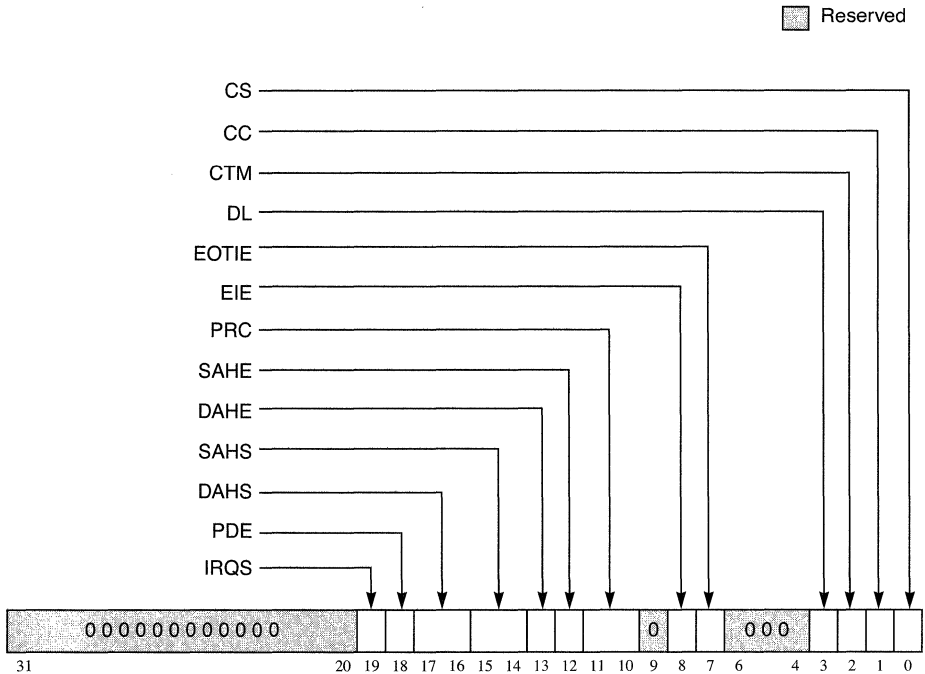


Figure 9-3. DMA Mode Register (DMR)

Table 9-3 describes the bit settings for the DMRs.

Table 9-3. DMR Field Descriptions—0x100

Bits	Name	Reset Value	R/W	Description
31–20	—	0	R	Reserved
19	IRQS	0	RW	Interrupt steer 0 Routes all DMA interrupts to the processor core through the internal $\overline{\text{int}}$ mechanism. 1 Routes all DMA interrupts to the PCI bus through the external $\overline{\text{INTA}}$ signal.

Table 9-3. DMR Field Descriptions—0x100 (Continued)

Bits	Name	Reset Value	R/W	Description
18	PDE	0	RW	Periodic DMA enable. Applies only to chaining mode. Otherwise, it is ignored. Refer to Section 9.3.2.2, "Periodic DMA Feature" for more information. 0 Disables periodic DMA restart 1 Allows hardware to periodically restart the DMA process.
17–16	DAHTS	00	RW	Destination address hold transfer size. Indicates the transfer size used for each transaction when the DAHE bit is set. The BCR value must be in multiples of this size and the DAR value must be aligned based on this size. 00 1 byte 01 2 bytes 10 4 bytes 11 8 bytes
15–14	SAHTS	00	RW	Source address hold transfer size. Indicates the transfer size used for each transaction when the SAHE bit is set. The BCR value must be in multiples of this size and the SAR value must be aligned based on this size. 00 1 byte 01 2 bytes 10 4 bytes 11 8 bytes
13	DAHE	0	RW	Destination address hold enable. Allows the DMA controller to hold the destination address to a fixed value for every transfer. The size used for the transfers is indicated by DAHTS. The MPC8240 only supports aligned transfers for this feature. Only one of DAHE or SAHE may be set at one time. 0 Disables the destination address hold feature 1 Enables the destination address hold feature
12	SAHE	0	RW	Source address hold enable. Allows the DMA controller to hold the source address to a fixed value for every transfer. The size used for the transfers is indicated by SAHTS. The MPC8240 only supports aligned transfers for this feature. Only one of DAHE or SAHE may be set at one time. 0 Disables the source address hold feature 1 Enables the source address hold feature
11–10	PRC	00	RW	PCI Read Command. Indicates the types of PCI read command to be used. 00 PCI Read 01 PCI Read Line 10 PCI Read Multiple 11 reserved
9	—	0	R	Reserved
8	EIE	0	RW	Error interrupt enable. Interrupt mechanism used depends on the setting of the IRQS bit. 0 Disables error interrupts 1 Generates an interrupt if there is a memory or PCI error during a DMA transfer (signalled by the setting of LME or PE in the DSR).
7	EOTIE	0	RW	End-of-transfer interrupt enable. Interrupt mechanism used depends on the setting of the IRQS bit. 0 Disables end-of-transfer interrupts 1 Generates an interrupt at the completion of a DMA transfer. (i.e. NDAR[EOTD] bit is set). For chained DMA, the interrupt is driven active at the end of the last segment. For periodic DMA, the interrupt is driven at the end of each periodic transfer event.

Table 9-3. DMR Field Descriptions—0x100 (Continued)

Bits	Name	Reset Value	R/W	Description
6-4	—	000	R	Reserved
3	DL	0	RW	Descriptor location. 0 The descriptor is located in the local memory space. 1 The descriptor is located in the PCI memory space.
2	CTM	0	RW	Channel transfer mode. 0 Chaining mode. See Section 9.3.2, “DMA Chaining Mode.” 1 Direct DMA mode. Thus, software is responsible for placing all the required parameters into the necessary registers to start the DMA process. See Section 9.3.1, “DMA Direct Mode.”
1	CC	0	RW	Channel continue. This bit applies only to chaining mode and is cleared by the MPC8240 after every descriptor read. 0 Channel stopped. 1 The DMA transfer will restart the transferring process starting at the current descriptor address (in CDAR).
0	CS	0	RW	Channel start. 0 to 1 transition when the channel is not busy (DSR[CB] = 0) starts the DMA process. If the channel is busy and a 0 to 1 transition occurs, then the DMA channel restarts from a previous halt condition. 1 to 0 transition when the channel is busy (DSR[CB] = 1) halts the DMA process. Nothing happens if the channel is not busy and a 1 to 0 transition occurs.

9.6.2 DMA Status Registers (DSRs)

The DSRs report various DMA conditions during and after the DMA transfer. Writing a 1 to a set bit clears the bit. Software attempting to determine the source of interrupts should always perform a logical AND function between the bits of the DSR and their corresponding enable bits in the DMR and CDAR. Figure 9-4 shows the bits in the DSRs.

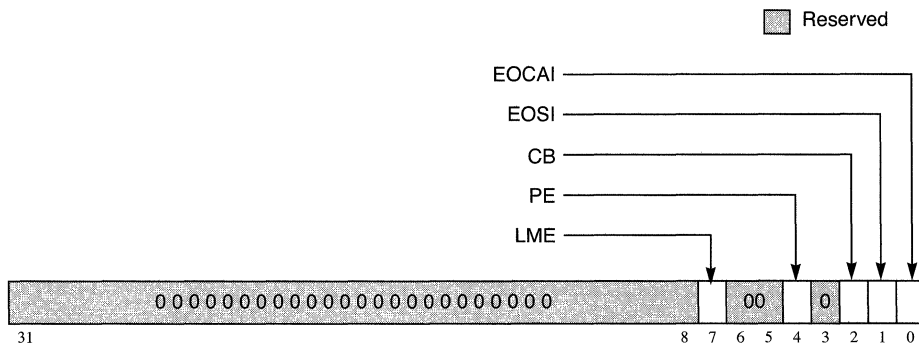


Figure 9-4. DMA Status Register (DSR)

Table 9-5 describes the bit settings for the DSRs.

Table 9-4. DSR Field Descriptions—0x104

Bit	Name	Reset Value	R/W	Description
31–8	—	All 0s	R	Reserved
7	LME	0	R/W Write1 clears	Local memory error. 0 No local memory error. When this bit is set, it can only be cleared by writing a 1 to it or by a hard reset. 1 A memory error condition occurred during the DMA transfer. If DMR[EIE] = 1, an interrupt is generated.
6–5	—	00	R	Reserved
4	PE	0	R/W Write1 clears	PCI error. 0 No PCI error. When this bit is set, it can only be cleared by writing a 1 to it or by a hard reset. 1 An abort condition or a parity error occurred on the PCI bus during the DMA transfer. If DMR[EIE] = 1, an interrupt is generated.
3	—	0	R	Reserved
2	CB	0	R	Channel busy. 0 Channel not busy. This bit is cleared by the MPC8240 as a result of an error or when the DMA transfer is finished. 1 A DMA transfer is currently in progress.
1	EOSI	0	R/W Write1 clears	End-of-segment interrupt. 0 No end-of-segment condition. When this bit is set, it can only be cleared by writing a 1 to it or by a hard reset. 1 If CDAR[EOSIE] = 1 and the block of data has finished transferring, this bit is set and an interrupt is generated.
0	EOCAI	0	R Write1 clears	End-of-chain/direct interrupt. 0 DMA transfer not finished. When this bit is set, it can only be cleared by writing a 1 to it or by a hard reset. 1 If DMR[EOTIE] = 1 and the last DMA transfer is finished, either in chaining or direct mode, this bit is set and an interrupt is generated.

9.6.3 Current Descriptor Address Registers (CDARs)

The CDARs contains the current address of the descriptor in memory to be loaded, one for each DMA channel. In chaining mode, software must initialize this register to point to the first descriptor in memory.

After finishing the first descriptor, if the EOTD bit in the next descriptor address register (NDAR) is not set, the DMA controller loads the contents of the NDAR into the CDAR and begins working on the next descriptor in memory. If the NDAR[EOTD] = 1, then the DMA transfer is finished. The SNEN, EOSIE, and CTT bits are used in both chaining and direct modes.

Figure 9-5 shows the bits in the CDARs.

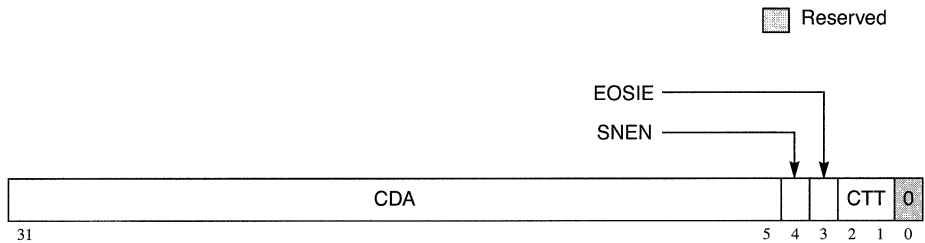


Figure 9-5. Current Descriptor Address Register (CDAR)

Table 9-5 describes the bit settings for the CDARs.

Table 9-5. CDAR Field Descriptions—0x108

Bits	Name	Reset Value	R/W	Description
31–5	CDA	All 0s	RW	Current descriptor address. Contains the current descriptor address of the buffer descriptor in memory. It must be aligned on an 8-word boundary. These bits are valid only for chaining mode.
4	SNEN	0	RW	Snoop enable. This bit is valid for both chaining and direct modes. 0 Disables snooping 1 Enables processor core snooping for DMA transactions.
3	EOSIE	0	RW	End-of-segment interrupt enable. Interrupt mechanism used depends on the setting of DMR[IRQS]. This bit is valid only for chaining mode. 0 End-of-segment interrupt disabled 1 Generates an interrupt if the DMA transfer for the current descriptor is finished.
2-1	CTT	00	RW	Channel transfer type. These two bits specify the type/direction of the DMA transfer. These bits are valid for both chaining and direct modes. 00 Local memory to local memory transfer 01 Local memory to PCI transfer 10 PCI to local memory transfer 11 PCI to PCI transfer
0	—	0	R	Reserved

9.6.4 Source Address Registers (SARs)

The SARs indicate the address from which the DMA controller reads data. This address can be either a PCI memory or local memory address. The software has to ensure that this is a valid memory address.

Figure 9-6 shows the bits in the SARs.

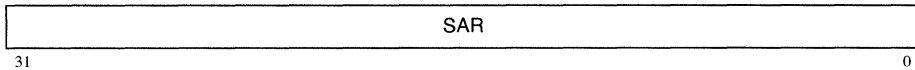


Figure 9-6. Source Address Register (SAR)

Table 9-5 describes the bit settings for the SARs.

Table 9-6. SAR Field Description—0x110

Bit	Name	Reset Value	R/W	Description
31-0	SAR	All 0s	RW	Source address. This register contains the source address of the DMA transfer. The content is updated by the MPC8240 after every DMA read operation.

9.6.5 Destination Address Registers (DARs)

The DARs indicate the address to which the DMA controller writes data. This address can be either a PCI memory or local memory address. The software has to ensure that this is a valid memory address.

Figure 9-7 shows the bits in the SARs.

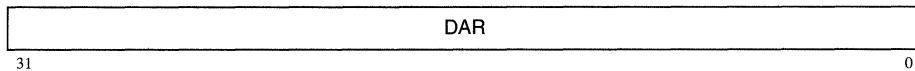


Figure 9-7. Destination Address Register (DAR)

Table 9-5 describes the bit settings for the DARs.

Table 9-7. DAR Field Description—0x118

Bit	Name	Reset Value	R/W	Description
31-0	DAR	All 0s	RW	Destination address. This register contains the destination address of the DMA transfer. The content is updated by the MPC8240 after every DMA write operation.

9.6.6 Byte Count Registers (BCRs)

The BCRs contain the number of bytes per transfer. The maximum transfer size is 64 Mbytes.

Figure 9-8 shows the bits in the BCRs.

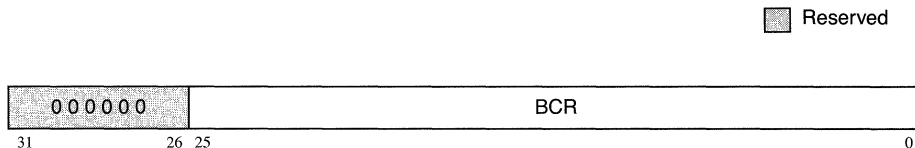


Figure 9-8. Byte Count Register (BCR)

Table 9-8 describes the bit settings for the BCRs.

Table 9-8. BCR Field Descriptions—0x120

Bit	Name	Reset Value	R/W	Description
31–26	—	All 0s	RW	Reserved
25–0	BCR	All 0s	RW	Byte count. Contains the number of bytes to transfer. The value in this register is automatically decremented by the MPC8240 after each DMA read operation until BCR = 0.

9.6.7 Next Descriptor Address Registers (NDARs)

The NDARs contain the address for the next descriptor in memory. All data bits, with the exception of EOTD, belong to the next descriptor to be loaded and executed. When the data bits are transferred to the CDAR, the bits become effective for the current transfer.

Figure 9-9 shows the bits in the NDARs.

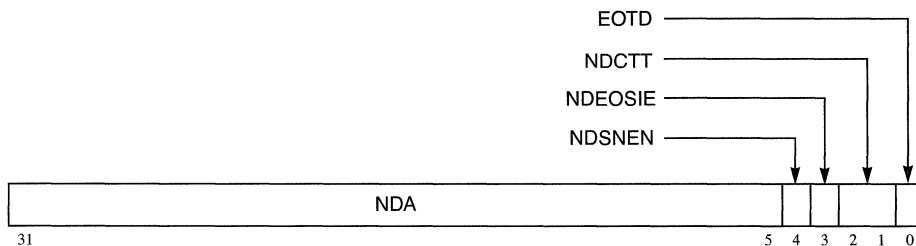


Figure 9-9. Next Descriptor Address Register (NDAR)

Table 9-5 describes the bit settings for the NDARs.

Table 9-9. NDAR Field Descriptions—0x124

Bit	Name	Reset Value	R/W	Description
31–5	NDA	All 0s	RW	Next descriptor address—Contains the next descriptor address of the buffer descriptor in memory; must be aligned on an 8-word boundary.
4	NDSNEN	0	RW	Next descriptor snoop enable—This bit is valid for both chaining and direct modes 0 Disables snooping 1 Enables processor core snooping for DMA transactions.
3	NDEOSIE	0	RW	Next descriptor end-of-segment interrupt enable—Interrupt mechanism used depends on the setting of DMR[IRQS]. This bit is valid only for chaining mode. 0 End-of-segment interrupt disabled 1 Generates an interrupt if the DMA transfer for the next descriptor is finished.
2–1	NDCTT	00	RW	Next descriptor channel transfer type—These two bits specify the type/direction of the DMA transfer. These bits are valid for both chaining and direct modes. 00 Local memory to local memory transfer 01 Local memory to PCI transfer 10 PCI to local memory transfer 11 PCI to PCI transfer
0	EOTD	0	RW	End-of-transfer descriptor—This bit is ignored in direct mode. 0 This descriptor is not the last descriptor in memory 1 Indicates that this descriptor is the last descriptor in memory. If this bit is set, the DMA controller will finish after the current buffer transaction is finished.

Chapter 10

Message Unit (with I₂O)

The MPC8240 provides a message unit (MU) to facilitate communications between the host processor and peripheral processors. The MPC8240's MU can operate with generic messages and doorbell registers; it also implements an I₂O-compliant interface. This chapter describes both the interfaces implemented by the MU and provides the details on the registers used by the MU.

10.1 Message Unit (MU) Overview

The embedded processor is often part of a larger system containing many processors and distributed memory. These processors tend to work on tasks independent of host processors 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. The MU of the MPC8240 provides the following features which can be used for this communication:

- A generic message and doorbell register interface
- A I₂O-compliant interface

10.2 Message and Doorbell Register Programming Model

The message and doorbell registers are described in the following subsections. Note that the interrupt status and interrupt mask bits for the message and doorbell registers are in the OMISR, OMIMR, IMISR, and IMIMR I₂O registers. See Section 10.3.4.1, "PCI-Accessible I₂O Registers;" and Section 10.3.4.2, "Processor-Accessible I₂O Registers."

10.2.1 Message and Doorbell Register Summary

MPC8240 contains two 32-bit inbound message registers (IMR0 and IMR1) and two 32-bit outbound message registers (OMR0 and OMR1).

Table 10-1 summarizes the message registers.

Table 10-1. Message Register Summary

PCI Offset	Local Memory Offset	Acronym	Name
0x050	0x0_0050	IMR0	Inbound message register 0
0x054	0x0_0054	IMR1	Inbound message register 1
0x058	0x0_0058	OMR0	Outbound message register 0
0x05C	0x0_005C	OMR1	Outbound message register 1

The MPC8240 contains a 32-bit inbound and a 32-bit outbound doorbell register (IDBR and ODBR, respectively). Table 10-2 summarizes the doorbell registers.

Table 10-2. Doorbell Register Summary

PCI Offset	Local Memory Offset	Acronym	Name
0x068	0x0_0068	IDBR	Inbound doorbell register
0x060	0x0_0060	ODBR	Outbound doorbell register

10.2.2 Message Register Descriptions

The IMRs allow a remote host or PCI master to write a 32-bit value, which automatically causes an interrupt to the processor core. The OMRs allow the processor core to write an outbound message that also automatically causes the outbound interrupt signal \overline{INTA} to be asserted. These interrupts can be masked in the IMIMR and OMIMR. When these registers are written, their corresponding interrupt status bits in the IMISR and OMISR are set. Figure shows the bits of the IMRs and OMRs.

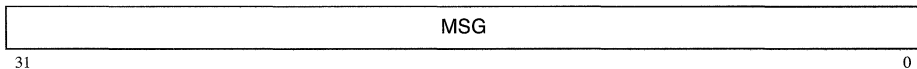


Figure 10-1. Message Registers (IMRs and OMRs)

Table 10-3 shows the bits settings for the IMRs and OMRs.

Table 10-3. IMR and OMR Field Descriptions— Offset 0x050–0x05C

Bits	Name	Reset Value	R/W	Description
31–0	MSG	Undefined	RW	The inbound and outbound message registers contain generic message data to be passed between the processor core and remote processors.

10.2.3 Doorbell Register Descriptions

The IDBR allows a remote processor to set a bit in the register from the PCI bus. This, in turn, causes an interrupt to be generated by the processor core. After the local interrupt is generated, it can only be cleared by the processor core by writing a 1 to the bits that are set in the IDBR. The remote processor can only generate the local interrupt through the IDBR; it can not clear the interrupt. Figure 10-2 shows the IDBR.

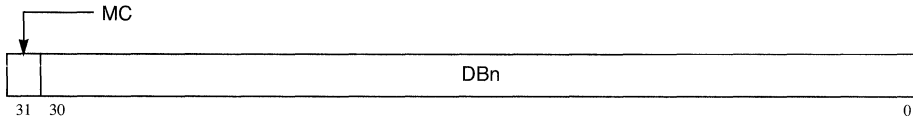


Figure 10-2. Inbound Doorbell Register (IDBR)

Table 10-4 shows the bit settings for the IDBR.

Table 10-4. IDBR Field Descriptions— Offset 0x068

Bits	Name	Reset Value	R/W	Description
31	MC	0	A write of 1 from PCI sets the bit; a write of 1 from the processor core clears the bit.	Machine check 0 No machine check 1 Writing to this bit generates a machine check exception to the processor core; it also causes IMISR[DMC] to be set.
30-0	DBn	All 0s	A write of 1 from PCI sets the bit; a write of 1 from the processor core clears the bit.	Inbound doorbell n interrupt, where n is each bit 0 No inbound doorbell interrupt 1 Setting any bit in this register from the PCI bus causes an interrupt to be generated through the $\overline{\text{INTA}}$ signal to the processor core; it also causes IMISR[IDI] to be set.

Alternatively, the MPC8240 processor core can write to the ODBR, which causes the outbound interrupt signal $\overline{\text{INTA}}$ to be asserted, thus interrupting the remote processor. When $\overline{\text{INTA}}$ is generated, it can only be cleared by the remote processor (through PCI) by writing a 1 to the bits that are set in the ODBR. The processor core can only generate $\overline{\text{INTA}}$ through the ODBR, and it can not clear this interrupt.

Figure 10-3 shows the ODBR.

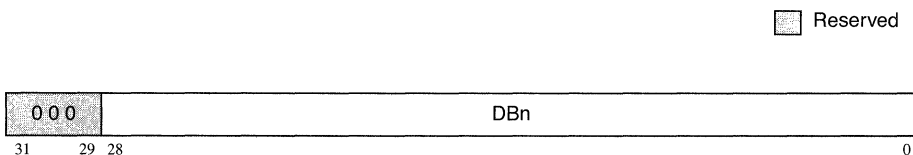


Figure 10-3. Outbound Doorbell Register (ODBR)

I₂O Interface

Table 10-5 shows the bit settings for the ODBR.

Table 10-5. ODBR Field Descriptions— Offset 0x060

Bits	Name	Reset Value	R/W	Description
31–29	—	000	R	Reserved
28–0	DBn	All 0s	A write of 1 from the processor core sets the bit; a write of 1 from PCI clears the bit.	Outbound doorbell interruptn where n is each bit. Writing any bit in this register from the processor core causes an external interrupt (INTA) to be signalled.

10.3 I₂O Interface

The intelligent input output (I₂O) specification was established in the industry to allow architecture-independent I/O subsystems to communicate with an OS through an abstraction layer. The specification is centered around a message-passing scheme. An I₂O-compliant embedded peripheral (IOP) is comprised of memory, processor, and input/output devices. The IOP dedicates a certain space in its local memory to hold inbound (from the remote processor) and outbound (to the remote processor) messages. The space is managed as memory-mapped FIFOs with pointers to this memory maintained through the MPC8240 I₂O registers.

10.3.1 PCI Configuration Identification

The I₂O specification defines extensions for the PCI bus through which message queues are managed in hardware. A host identifies an IOP by its PCI class code. Table 10-6 provides the configuration information available to the host when the I₂O unit is enabled.

Table 10-6. I₂O PCI Configuration Identification Register Settings

PCI Configuration Offset	Local Memory Offset	Register	Description
0x0B	0x0B	PCI CFG	PCI base class—PCI configuration data returned: 0x0E
0x0A	0x0A	PCI CFG	Sub Class—PCI configuration PCI data returned: 0x00
0x09	0x09	PCI CFG	Programming interface code—PCI configuration PCI data returned: 0x01

See Section 5.2, “PCI Interface Configuration Registers,” for more information on the PCI base class and programming interface codes. The subclass is described in more detail in the PCI specification.

10.3.2 I₂O Register Summary

The MPC8240 I₂O registers are summarized in Table 10-7.

Table 10-7. I₂O Register Summary

PCI Offset	Local Memory Offset	Acronym	Name
0x030	—	OMISR	Outbound message interrupt status register
0x034	—	OMIMR	Outbound message interrupt mask register
0x040	—	IFQPR	Inbound FIFO queue port register
0x044	—	OFQPR	Outbound FIFO queue port register
—	0x0_0100	IMISR	Inbound message interrupt status register
—	0x0_0104	IMIMR	Inbound message interrupt mask register
—	0x0_0120	IFHPR	Inbound free_FIFO head pointer register
—	0x0_0128	IFTPR	Inbound free_FIFO tail pointer register
—	0x0_0130	IPHPR	Inbound post_FIFO head pointer register
—	0x0_0138	IPTPR	Inbound post_FIFO tail pointer register
—	0x0_0140	OFHPR	Outbound free_FIFO head pointer register
—	0x0_0148	OFTPR	Outbound free_FIFO tail pointer register
—	0x0_0150	OPHPR	Outbound post_FIFO head pointer register
—	0x0_0158	OPTPR	Outbound post_FIFO tail pointer register
—	0x0_0164	MUCR	Messaging unit control register
—	0x0_0170	QBAR	Queue base address register. Must be set on 1-Mbyte boundary

10.3.3 FIFO Descriptions

There are two paths for messages—an inbound queue to receive messages from the remote host (and other IOPs) and an outbound queue to pass messages to a remote host. Each queue is implemented as a pair of FIFOs. The inbound and outbound message queues each consists of a free_list FIFO and a post_list FIFO.

Messages are made up of frames which are at least 64 bytes long. The message frame address (MFA) points to the first byte of the message frame. The messages are located in a pool of system memory (any memory address accessible through the PCI bus). Tracking of the status and location of these messages is done with the four FIFOs, also in local memory. One FIFO in each queue tracks the free MFAs (free_list FIFO). The other FIFO tracks the MFAs that have posted messages (post_list FIFO). These FIFOs are managed by the remote processors and the processor core through the MPC8240 I₂O registers. For more information, see Section 10.3.2, “I₂O Register Summary”.

Figure 10-4 shows an example of the message queues:

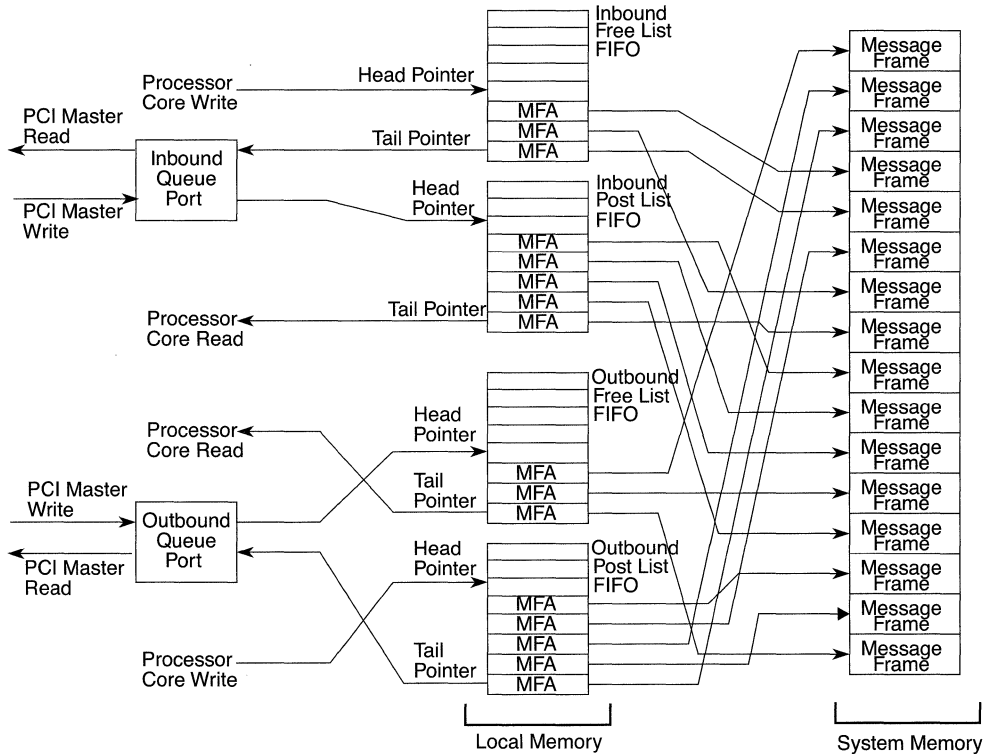


Figure 10-4. I₂O Message Queue Example

Table 10-8 lists the queue starting addresses for the FIFOs.

Table 10-8. Queue Starting Address

FIFO	Starting Address
Inbound free_list	QBA (specified in QBAR)
Inbound post_list	QBA + (1*FIFO size specified in MUCR)
Outbound post_list	QBA + (2*FIFO size specified in MUCR)
Outbound free_list	QBA + (3*FIFO size specified in MUCR)

The following subsections describe the inbound and outbound FIFOs of the I₂O interface.

10.3.3.1 Inbound FIFOs

The I₂O specification defines two inbound FIFOs—an inbound post_list FIFO and an inbound free_list FIFO. The inbound FIFOs allow external PCI masters to post messages to the MPC8240 processor core.

10.3.3.1.1 Inbound Free_List FIFO

The inbound free_list FIFO holds the list of empty inbound MFAs. The external PCI master reads the inbound FIFO queue port register (IFQPR) which returns the MFA pointed to by the inbound free_FIFO tail pointer register (IFTPR). The MPC8240's I₂O unit then automatically increments the value in IFTPR.

If the inbound free_list FIFO is empty (no free MFA entries), the unit returns 0xFFFF_FFFF.

10.3.3.1.2 Inbound Post_List FIFO

The inbound post_list FIFO holds MFAs that are posted to the processor core from external PCI masters. PCI masters external to MPC8240 write to the head of the FIFO by writing the MFA to the inbound FIFO queue port register (IFQPR). The I₂O unit transfers the MFA to the location pointed to by the inbound post_FIFO head pointer register (IPHPR).

After the data is written to local memory, the MPC8240's I₂O unit automatically increments the value in IPHPR to set up for the next message. In addition, an interrupt is generated to the processor core (provided the interrupt is not masked). The inbound post queue interrupt bit in the inbound message interrupt status register (IMISR[IPQI]) is set to indicate the condition. The processor core should clear the interrupt bit as part of the interrupt handler and can read the message pointed to by the MFA located in the IPTPR. After the message has been read, the interrupt software must explicitly increment the value in IPTPR.

When the processor is done using the message, it must return the message to the inbound free_list FIFO.

10.3.3.2 Outbound FIFOs

The I₂O specification defines two outbound FIFOs—an outbound post_list FIFO and an outbound free_list FIFO. The outbound FIFOs are used to send messages from the processor core to a remote host processor.

10.3.3.2.1 Outbound Free_List FIFO

The outbound free_list FIFO holds the MFAs of the empty outbound message locations in local memory. When the processor core is ready to send an outbound message, first, it gets an empty MFA by reading the OFTPR; then it writes the message into the MFA. The OFTPR is managed by the processor core.

When an external PCI master is done using a message posted in the outbound post_list FIFO and wants to return the MFA to the free list, it writes to the outbound FIFO queue port register (OFQPR). The MPC8240 I₂O unit then writes the MFA to the outbound free_FIFO head pointer register (OFHPR). This, in turn causes the value in OFHPR to automatically be incremented.

10.3.3.2.2 Outbound Post_List FIFO

The outbound post_list FIFO holds MFAs that are posted from the processor core to remote processors. The processor core places messages in the outbound post_list FIFO by writing the MFA to OPHPR. This software must then increment the value in OPHPR.

When the FIFO is not empty (head and tail pointers are not equal), the outbound post_list queue interrupt bit in the outbound message interrupt status register, OMISR[OPQI], is set. Additionally, the external MPC8240 PCI interrupt ($\overline{\text{INTA}}$) is asserted (if it is not masked). When the head and tail pointers are equal, OMISR[OPQI] is cleared. The outbound post_list queue interrupt can be masked using the outbound message interrupt mask register (OMIMR).

An external PCI master reads the outbound FIFO queue port register (OFQPR) to cause the MPC8240's I₂O unit to read the MFA from local memory pointed to by the OPTPR. The unit then automatically increments the OPTPR.

When the FIFO is empty (head and tail pointers are equal), the unit returns 0xFFFF_FFFF.

10.3.4 I₂O Register Descriptions

The following sections provide the detailed descriptions of the I₂O registers (and some of the bits that control the generic message and doorbell register interface in these registers).

10.3.4.1 PCI-Accessible I₂O Registers

The OMISR, OMIMR, IFQPR, and OFQPR registers are used by PCI masters to access the MPC8240 I₂O unit. The processor core cannot access any of these registers. See Chapter 12, "Embedded Programmable Interrupt Controller (EPIC)," for more information on the interrupt mechanisms of the MPC8240.

10.3.4.1.1 Outbound Message Interrupt Status Register (OMISR)

The OMISR contains the interrupt status of the I₂O, doorbell and outbound message registers. These events are generated by the MPC8240 and the interrupts are signalled to the PCI bus through the $\overline{\text{INTA}}$ signal. Writing a 1 to a set bit in OMISR clears the bit. Software attempting to determine the source of the interrupts should always perform a logical AND between the OMISR bits and their corresponding mask bits in the OMIMR.

Figure 10-5 shows the bits of the OMISR.

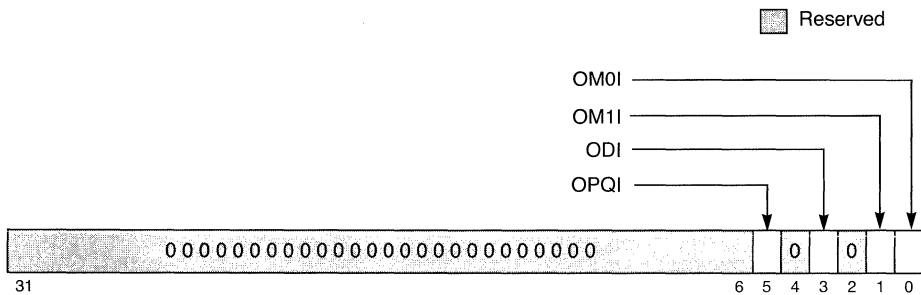


Figure 10-5. Outbound Message Interrupt Status Register (OMISR)

Table 10-9 shows the bit settings for the OMISR.

Table 10-9. OMISR Field Descriptions— Offset 0x030

Bits	Name	Reset Value	R/W	Description
31-6	—	All 0s	R	Reserved
5	OPQI	0	Read-only	Outbound post queue interrupt (I ₂ O interface) 0 No messages in the outbound queue—To clear this bit, software has to read all the MFAs in the outbound post_list FIFO. 1 Indicates that a message or messages are posted to the outbound post_list FIFO through the OFQPR. This bit is set independently of the outbound post queue interrupt mask (OMIMR[OPQIM]) bit.
4	—	0	R	Reserved
3	ODI	0	Read-only	Outbound doorbell interrupt 0 No outbound doorbell interrupt—This bit is automatically cleared when all bits in the ODBR are cleared. 1 Indicates an outbound doorbell interrupt condition (a bit in ODBR is set). Set independently of the mask bit in OMIMR.
2	—	0	R	Reserved
1	OM1I	0	Read; write 1 clears this bit	Outbound message 1 interrupt 0 No outbound message 1 interrupt 1 Indicates an outbound message 1 interrupt condition (a write occurred to OMR1). Set independently of the mask bit in OMIMR.
0	OM0I	0	Read; write 1 clears this bit	Outbound message 0 interrupt 0 No outbound message 0 interrupt 1 Indicates an outbound message 0 interrupt condition (a write occurred to OMR0). Set independently of the mask bit in OMIMR.

10.3.4.1.2 Outbound Message Interrupt Mask Register (OMIMR)

The OMIMR contains the interrupt masks of the I₂O, doorbell, and message register events generated by the MPC8240.

Figure 10-6 shows the bits of the OMIMR.

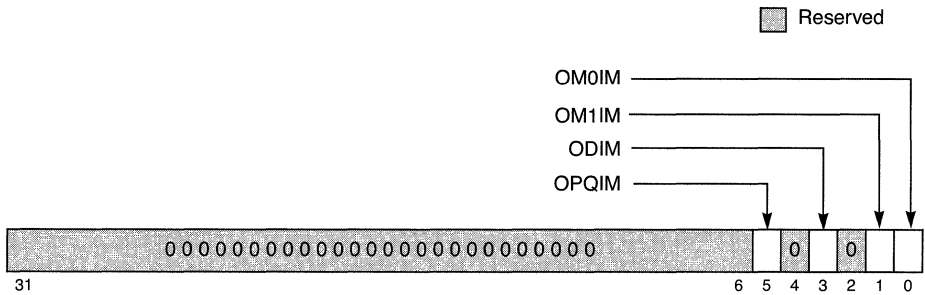


Figure 10-6. Outbound Message Interrupt Mask Register (OMIMR)

Table 10-10 shows the bit settings for the OMIMR.

Table 10-10. OMIMR Field Descriptions— Offset 0x034

Bits	Name	Reset Value	R/W	Description
31–6	—	All 0s	R	Reserved
5	OPQIM	0	RW	Outbound post queue interrupt mask 0 Outbound post queue interrupt is allowed. 1 Outbound post queue interrupt is masked.
4	—	0	R	Reserved
3	ODIM	0	RW	Outbound doorbell interrupt mask 0 Outbound doorbell interrupt is allowed. 1 Outbound doorbell interrupt is masked.
2	—	0	R	Reserved
1	OM1IM	0	RW	Outbound message 1 interrupt mask 0 Outbound message 1 interrupt is allowed. 1 Outbound message 1 interrupt is masked.
0	OM0IM	0	RW	Outbound message 0 interrupt mask 0 Outbound message 0 interrupt is allowed. 1 Outbound message 0 interrupt is masked.

10.3.4.1.3 Inbound FIFO Queue Port Register (IFQPR)

The IFQPR is used by PCI masters to access inbound messages in local memory. Figure 10-7 shows the bits of the IFQPR.

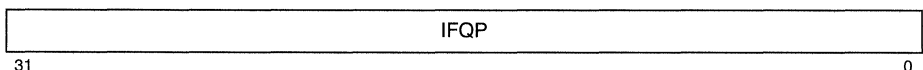


Figure 10-7. Inbound FIFO Queue Port Register (IFQPR)

Table 10-11 shows the bit settings for the IFQPR.

Table 10-11. IFQPR Field Descriptions— Offset 0x040

Bits	Name	Reset Value	R/W	Description
31–0	IFQP	All 0s	RW	Inbound FIFO queue port—Reading this register returns the MFA from the inbound free_list FIFO. Writing to this register posts the MFA to the inbound post_list FIFO.

10.3.4.1.4 Outbound FIFO Queue Port Register (OFQPR)

The OFQPR is used by PCI masters to access outbound messages in local memory. Figure 10-8 shows the bits of the OFQPR

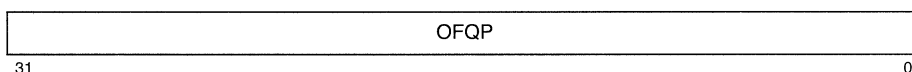


Figure 10-8. Outbound FIFO Queue Port Register (OFQPR)

Table 10-12 shows the bit settings for the OFQPR.

Table 10-12. OFQPR Field Descriptions— Offset 0x044

Bits	Name	Reset Value	R/W	Description
31–0	OFQP	All 0s	RW	Outbound FIFO queue port—Reading this register returns the MFA from the outbound post_list FIFO. Writing to this register posts the MFA to the outbound free_list FIFO.

10.3.4.2 Processor-Accessible I₂O Registers

The following sections describe the I₂O registers accessible by the processor core (and some of the bits that control the generic message and doorbell register interface in these registers).

10.3.4.2.1 Inbound Message Interrupt Status Register (IMISR)

The IMISR contains the interrupt status of the I₂O, doorbell, and message register events. These events are generated by the PCI masters and are routed to the processor core from the MU through the internal *int* or *mcp* signals as described in Table 10-13. See Chapter 12, “Embedded Programmable Interrupt Controller (EPIC),” for more information about the assertion of *int*; and Chapter 13, “Error Handling and Exceptions,” for more information about the enabling of machine check exceptions to the processor core.

Writing a 1 to a set bit in IMISR clears the bit. The MPC8240 software must service these interrupts and clear these interrupt bits. Software attempting to determine the source of the interrupts should always perform a logical AND between the IMISR bits and their corresponding mask bits in the IMIMR.

Figure 10-9 shows the bits of the IMISR.

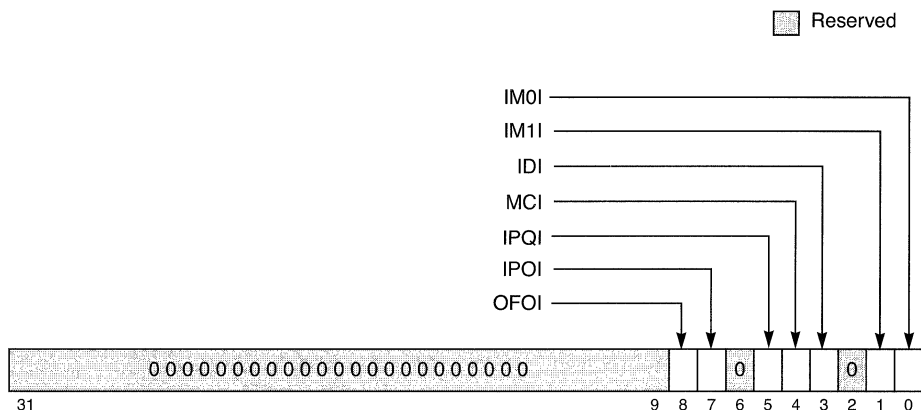


Figure 10-9. Inbound Message Interrupt Status Register (IMISR)

Table 10-13 shows the bit settings for the IMISR.

Table 10-13. IMISR Field Descriptions— Offset 0x0_0100

Bits	Name	Reset Value	R/W	Description
31–9	—	All 0s	R	Reserved
8	OFO	0	Read; write 1 clears this bit	Outbound free_list overflow condition 0 No overflow condition 1 Indicates that the outbound free_list FIFO head pointer is equal to the outbound free_list FIFO tail pointer and the queue is full. A machine check is signalled to the processor core through the internal mcp signal and a machine check exception is taken (if enabled). See Chapter 13, “Error Handling and Exceptions.” Set independently of the mask bit in IMIMR.
7	IPO	0	Read; write 1 clears this bit	Inbound post_list overflow condition 0 No overflow condition 1 Indicates that the inbound free_list FIFO head pointer is equal to the inbound free_list FIFO tail pointer and the queue is full. A machine check is signalled to the processor core through the internal mcp signal and a machine check exception is taken (if enabled). Set independently of the mask bit in IMIMR.
6	—	0	R	Reserved
5	IPQI	0	Read; write 1 clears this bit	Inbound post queue interrupt (I ₂ O interface) 0 No MFA in the IFQPR. 1 Indicates that the PCI master has posted an MFA to the inbound post_list FIFO through the IFQPR. Interrupt is signalled to the processor core through the internal int signal. This bit is set independently of the inbound post queue interrupt mask (IMIMR[IPQIM]) bit.

Table 10-13. IMISR Field Descriptions— Offset 0x0_0100 (Continued)

Bits	Name	Reset Value	R/W	Description
4	DMC	0	Read-only	Doorbell register machine check condition. 0 No doorbell register machine check condition. 1 Indicates that a remote processor has generated a machine check condition (causing assertion of mcp) by setting IDBR[MC]. This bit is cleared when IDBR[MC] is cleared. Note that this bit is not set if the mask bit, IMIMR[DMCM], = 1.
3	IDI	0	Read-only	Inbound doorbell interrupt 0 No inbound doorbell interrupt. This bit is cleared when the processor core clears IDBR[30–0]. 1 Indicates that at least one of IDBR[30–0] is set. Interrupt is signalled to the processor core through the internal int̄ signal. Set independently of the mask bit in IMIMR.
2	—	0	R	Reserved
1	IM1I	0	Read; write 1 clears this bit	Inbound message 1 interrupt. 0 No inbound message 1 interrupt. 1 Indicates an inbound message 1 interrupt condition (a write occurred to IMR1 from a remote PCI master). Interrupt is signalled to the processor core through the internal int̄ signal. Set independently of the mask bit in IMIMR.
0	IM0I	0	Read; write 1 clears this bit	Inbound message 0 interrupt. 0 No inbound message 0 interrupt. 1 Indicates an inbound message 0 interrupt condition (a write occurred to IMR0 from a remote PCI master). Interrupt is signalled to the processor core through the internal int̄ signal. Set independently of the mask bit in IMIMR.

10.3.4.2.2 Inbound Message Interrupt Mask Register (IMIMR)

The IMIMR contains the interrupt mask of the I₂O, doorbell, and message register events generated by a remote PCI master. Figure 10-10 shows the bits of the IMIMR.

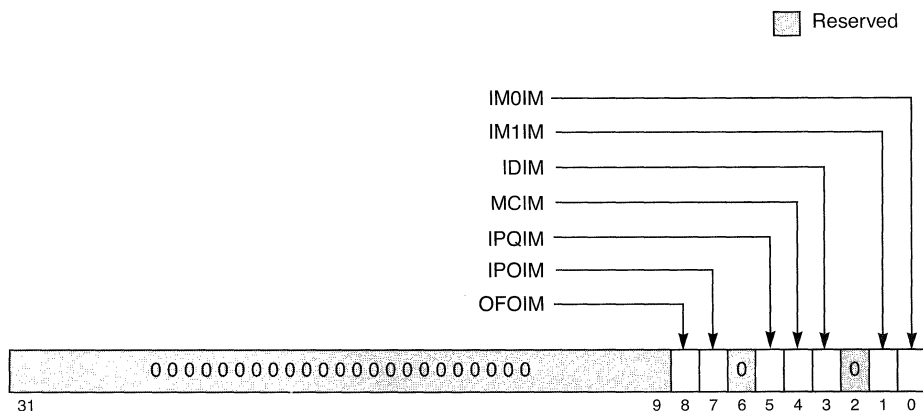


Figure 10-10. Inbound Message Interrupt Mask Register (IMIMR)

Table 10-14 shows the bit settings for the IMIMR.

Table 10-14. IMIMR Field Descriptions— Offset 0x0_0104

Bits	Name	Reset Value	R/W	Description
31–9	—	All 0s	R	Reserved.
8	OFOM	0	RW	Outbound free_list overflow mask 0 Outbound free_list overflow is allowed (and causes \overline{mcp}). 1 Outbound free_list overflow is masked.
7	IPOM	0	RW	Inbound post_list overflow mask 0 Inbound post_list overflow is allowed (and causes \overline{mcp}). 1 Inbound post_list overflow is masked.
6	—	0	R	Reserved.
5	IPQIM	0	RW	Inbound post queue interrupt mask 0 Inbound post queue interrupt is allowed. 1 Inbound post queue interrupt is masked.
4	DMCM	0	RW	Doorbell register machine check mask. 0 Doorbell machine check from IDBR[MC] is allowed. 1 Doorbell machine check from IDBR[MC] is masked. When this machine check condition is masked, the IMISR[DMC] is not set, regardless of the state of IDBR[MC].
3	IDIM	0	RW	Inbound doorbell interrupt mask 0 Inbound doorbell interrupt is allowed. 1 Inbound doorbell interrupt is masked.
2	—	0	R	Reserved
1	IM1IM	0	RW	Inbound message 1 interrupt 0 Inbound message 1 interrupt is allowed. 1 Inbound message 1 interrupt is masked.
0	IM0IM	0	RW	Inbound message 0 interrupt 0 Inbound message 0 interrupt is allowed. 1 Inbound message 0 interrupt is masked.

10.3.4.2.3 Inbound Free_FIFO Head Pointer Register (IFHPR)

Free MFAs are posted by the processor core to the inbound free_list FIFO pointed to by the inbound free_FIFO head pointer register (IFHPR). The processor core is responsible for updating the contents of IFHPR. Figure 10-11 shows the bits of the IFHPR.

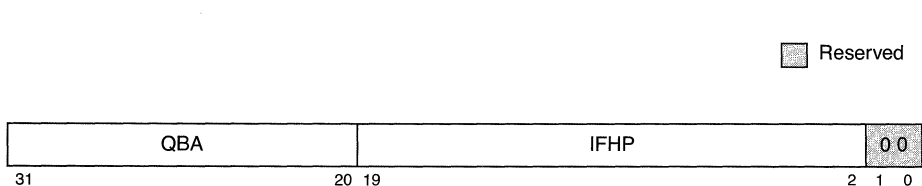


Figure 10-11. Inbound Free_FIFO Head Pointer Register (IFHPR)

Table 10-15 shows the bit settings for the IFHPR.

Table 10-15. IFHPR Field Descriptions— Offset 0x0_0120

Bits	Name	Reset Value	R/W	Description
31–20	QBA	All 0s	R	Queue base address—When read, this field returns the content of QBAR[31–20].
19–2	IFHP	All 0s	RW	Inbound free_FIFO head pointer—The processor maintains the local memory offset of the head pointer of the inbound free_list FIFO in this field.
1–0	—	00	R	Reserved

10.3.4.2.4 Inbound Free_FIFO Tail Pointer Register (IFTPR)

PCI masters pick up free MFAs from the inbound free_list FIFO pointed to by the inbound free_FIFO tail pointer register (IFTPR). The actual PCI reads of MFAs are performed through the inbound FIFO queue port register (IFQPR). The MPC8240 automatically increments the IFTP value after every read from IFQPR. Figure 10-12 shows the bits of the IFTPR.

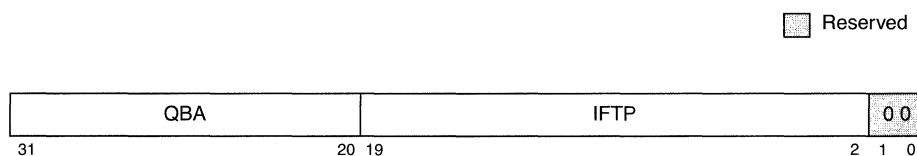


Figure 10-12. Inbound Free_FIFO Tail Pointer Register (IFTPR)

Table 10-16 shows the bit settings for the IFTPR.

Table 10-16. IFTPR Field Descriptions— Offset 0x0_0128

Bits	Name	Reset Value	R/W	Description
31–20	QBA	All 0s	R	Queue base address—When read, this field returns the content of QBAR[31–20].
19–2	IFTP	All 0s	RW	Inbound free_FIFO tail pointer—Maintains the local memory offset of the tail pointer of the inbound free_list FIFO.
1–0	—	00	R	Reserved

10.3.4.2.5 Inbound Post_FIFO Head Pointer Register(IPHPR)

PCI masters post MFAs to the inbound post_list FIFO pointed to by the inbound post_FIFO head pointer register (IPHPR). The actual PCI writes are performed through the inbound FIFO queue port register (IFQPR). The MPC8240 automatically increments the IPHP value after every write to IFQPR. Figure 10-13 shows the bits of the IPHPR.

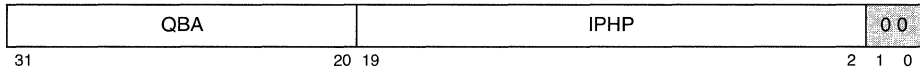
 Reserved


Figure 10-13. Inbound Post_FIFO Head Pointer Register (IPHPR)

Table 10-17 shows the bit settings for the IPHPR.

Table 10-17. IPHPR Field Descriptions— Offset 0x0_0130

Bits	Name	Reset Value	R/W	Description
31–20	QBA	All 0s	R	Queue base address—When read, this field returns the content of QBAR[31–20].
19–2	IPHP	All 0s	RW	Inbound post_FIFO head pointer—Maintains the local memory offset of the head pointer of the inbound post_list FIFO.
1–0	—	00	R	Reserved

10.3.4.2.6 Inbound Post_FIFO Tail Pointer Register(IPTPR)

The processor picks up MFAs posted by PCI masters from the inbound post_list FIFO pointed to by the inbound post_FIFO tail pointer register (IPTPR). The processor core is responsible for updating the contents of IPTPR. Figure 10-14 shows the bits of the IPTPR.

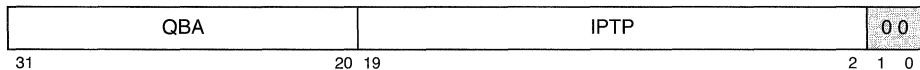
 Reserved


Figure 10-14. Inbound Post_FIFO Tail Pointer Register (IPTPR)


Table 10-18 shows the bit settings for the IPTPR.

Table 10-18. IPTPR Field Descriptions— Offset 0x0_0138

Bits	Name	Reset Value	R/W	Description
31–20	QBA	All 0s	R	Queue base address—When read, this field returns the content of QBAR[31–20].
19–2	IPTP	All 0s	RW	Inbound post_FIFO tail pointer—The processor maintains the local memory offset of the inbound post_list FIFO tail pointer in this field.
1–0	—	00	R	Reserved

10.3.4.2.7 Outbound Free_FIFO Head Pointer Register (OFHPR)

PCI masters return free MFAs to the inbound free_list FIFO pointed to by the inbound free_FIFO head pointer register (IFHPR). The actual PCI writes of MFAs are performed through the outbound FIFO queue port register (OFQPR). The MPC8240 automatically increments the OFTP value after every read from OFQPR. Figure 10-15 shows the bits of the OFHPR.

 Reserved

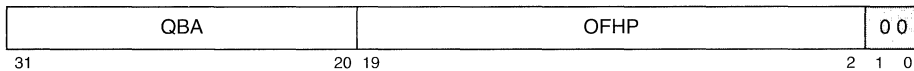


Figure 10-15. Outbound Free_FIFO Head Pointer Register (OFHPR)

Table 10-19 shows the bit settings for the OFHPR

Table 10-19. OFHPR Field Descriptions— Offset 0x0_0140

Bits	Name	Reset Value	R/W	Description
31–20	QBA	All 0s	R	Queue base address—When read, this field returns the content of QBAR[31–20].
19–2	OFHP	All 0s	RW	Outbound free_FIFO head pointer—Maintains the local memory offset of the head pointer of the outbound free_list FIFO.
1–0	—	0	R	Reserved

10.3.4.2.8 Outbound Free_FIFO Tail Pointer Register(OFTPR)

The processor picks up free MFAs from the outbound free_list FIFO pointed to by the outbound free_FIFO tail pointer register (OFTPR). The processor core is responsible for updating the contents of OFTPR. Figure 10-16 shows the bits of the OFTPR.

 Reserved

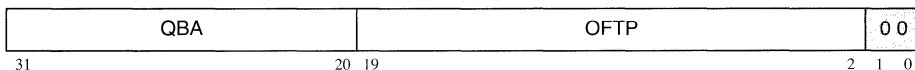


Figure 10-16. Outbound Free_FIFO Tail Pointer Register (OFTPR)

Table 10-20 shows the bit settings for the OFTPR.

Table 10-20. OFTPR Field Descriptions— Offset 0x0_0148

Bits	Name	Reset Value	R/W	Description
31–20	QBA	All 0s	R	Queue base address. When read, this field returns the content of QBAR[31–20].
19–2	OFTP	All 0s	RW	Outbound free_FIFO tail pointer. The processor maintains the local memory offset of the tail pointer of the outbound free_list FIFO in this field.
1–0	—	00	R	Reserved

10.3.4.2.9 Outbound Post_FIFO Head Pointer Register(OPHPR)

The processor core posts MFAs to the outbound post_list FIFO pointed to by the outbound post_FIFO head pointer register (OPHPR). The processor core is responsible for updating the contents of OPHPR. Figure 10-17 shows the bits of the OPHPR.

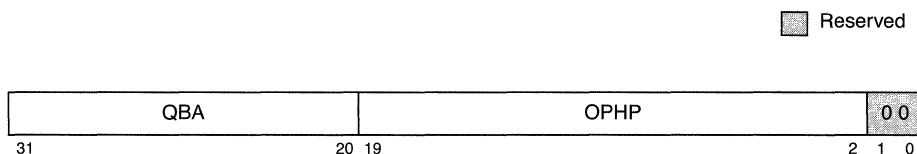


Figure 10-17. Outbound Post_FIFO Head Pointer Register (OPHPR)

Table 10-21 shows the bit settings for the OPHPR.

Table 10-21. OPHPR Field Descriptions— Offset 0x0_0150

Bits	Name	Reset Value	R/W	Description
31–20	QBA	All 0s	R	Queue base address—When read, this field returns the content of QBAR[31–20].
19–2	OPHP	All 0s	RW	Outbound post_FIFO head pointer—The processor maintains the local memory offset of the head pointer of the outbound post_list FIFO in this field.
1–0	—	00	R	Reserved

10.3.4.2.10 Outbound Post_FIFO Tail Pointer Register (OPTPR)

PCI masters pick up posted MFAs from the outbound post_list FIFO pointed to by the outbound post_FIFO tail pointer register (OPTPR). The actual PCI reads of MFAs are performed through the outbound FIFO queue port register (OFQPR). The MPC8240 automatically increments the OPTP value after every read from OFQPR.

Figure 10-18 shows the bits of the OPTPR.

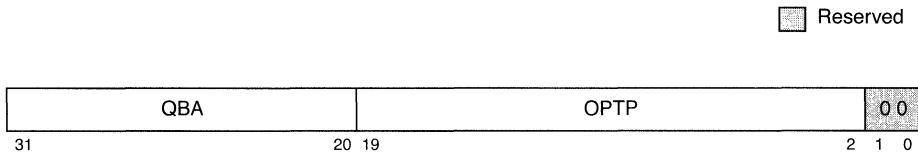


Figure 10-18. Outbound Post_FIFO Tail Pointer Register (OPTPR)

Table 10-22 shows the bit settings for the OPTPR.

Table 10-22. OPTPR Field Descriptions— Offset 0x0_0158

Bits	Name	Reset Value	R/W	Description
31–20	QBA	All 0s	R	Queue base address—When read, this field returns the content of QBAR[31–20].
19–2	OPTP	All 0s	RW	Outbound post_FIFO tail pointer—Maintains the local memory offset of the tail pointer of the outbound post_list FIFO.
1–0	—	00	R	Reserved

10.3.4.2.11 Messaging Unit Control Register (MUCR)

The MUCR allows software to enable and set up the size of the inbound and outbound FIFOs. Figure 10-19 shows the bits of the MUCR.

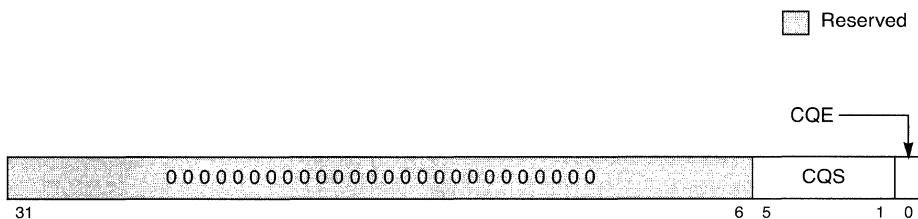


Figure 10-19. Messaging Unit Control Register (MUCR)

Table 10-23 shows the bit settings for the MUCR

Table 10-23. MUCR Field Descriptions— Offset 0x0_0164

Bits	Name	Reset Value	R/W	Description
31–6	—	All 0s	R	Reserved
5–1	CQS	0b0_0001	RW	Circular queue size 0b0_0001: 4K entries (16 Kbytes) 0b0_0010: 8K entries (32 Kbytes) 0b0_0100: 16K entries (64 Kbytes) 0b0_1000: 32K entries (128 Kbytes) 0b1_0000: 64K entries (256 Kbytes)
0	CQE	0	RW	Circular queue enable 0 PCI writes to IFQPR and OFQPR are ignored and reads return 0xFFFF_FFFF. 1 Allows PCI masters to access the inbound and outbound queue ports (IFQPR and OFQPR). Normally, this bit is set only after software has initialized all pointers and configuration registers.

10.3.4.2.12 Queue Base Address Register (QBAR)

The QBAR specifies the beginning address of the circular queue structure in local memory. Figure 10-20 shows the bits of the QBAR.

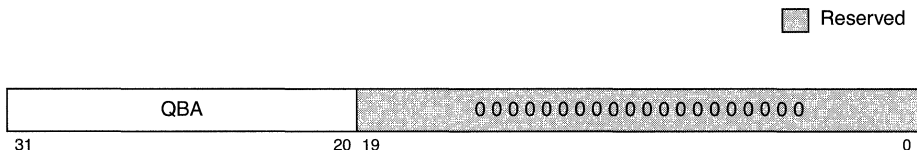


Figure 10-20. Queue Base Address Register (QBAR)

Table 10-24 shows the bit settings for the QBAR.

Table 10-24. QBAR Field Descriptions— Offset 0x0_0170

Bits	Name	Reset Value	R/W	Description
31–20	QBA	All 0s	RW	Queue base address. Base address of circular queue in local memory. Note that the circular queue must be aligned on a 1Mbyte boundary.
19–0	—	All 0s	R	Reserved

Chapter 11

I²C Interface

This chapter describes the I²C (inter-integrated circuit) interface on the MPC8240.

11.1 I²C Interface Overview

The I²C interface is a two-wire, bidirectional serial bus developed by Philips that provides a simple, efficient way to exchange data between integrated circuit (IC) devices. The I²C interface allows the MPC8240 to exchange data with other I²C devices such as microcontrollers, EEPROMs, real-time clock devices, A/D converters, and LCDs. The two-wire bus—serial data and serial clock—minimizes device interconnections. The synchronous, multimaster bus of the I²C allows the connection of additional devices to the bus for expansion and system development.

The I²C interface is a true multimaster bus that includes collision detection and arbitration that prevent data corruption if two or more masters attempt to control the bus simultaneously. This feature allows for complex applications with multiprocessor control.

11.1.1 I²C Unit Features

The I²C unit on the MPC8240 consists of a transmitter/receiver unit, a clocking unit, and a control unit. Some of the features of the I²C unit are as follows:

- Two-wire interface
- Multimaster support
- Master or slave I²C mode support
- Software-programmable for one of 64 different serial clock frequencies
- Software-selectable acknowledge bit
- Interrupt-driven, byte-to-byte data transfer
- Arbitration-lost interrupt with automatic mode switching from master to slave
- Calling address identification interrupt
- START and STOP condition generation/detection
- Repeated START condition generation
- Acknowledge bit generation/detection
- Bus-busy detection
- Programmable on-chip digital filter rejects electrical spikes on the bus

11.1.2 I²C Interface Signal Summary

The I²C interface uses the serial data (SDA) signal and serial clock (SCL) signal for data transfer. All devices connected to these two signals must have open-drain or open-collector outputs. A logical AND function is performed on both signals with external pull-up resistors.

Table 11-1 summarizes the two signals that comprise the I²C interface.

Table 11-1. I²C Interface Signal Description

Signal Name	Idle State	I/O	State Meaning
SCL (serial clock)	HIGH	I	When the MPC8240 is idle or acts as a slave, SCL defaults as an input. The unit uses SCL to synchronize incoming data on SDA. The bus is assumed to be busy when SCL is detected low.
		O	As a master, the MPC8240 drives SCL along with SDA when transmitting. As a slave, the MPC8240 drives SCL low for data pacing.
SDA (serial data)	HIGH	I	When the MPC8240 is idle or in a receiving mode, SDA defaults as an input. The unit receives data from other I ² C devices on SDA. The bus is assumed to be busy when SDA is detected low.
		O	As a master or slave, the MPC8240 drives data on SDA synchronous to SCL.

11.1.3 I²C Block Diagram

The reset state of the I²C interface is as a slave receiver. Thus, when not explicitly programmed to be a master or to respond to a slave transmitter address, the I²C unit always defaults to a slave receiver operation. Figure 11-1 shows a block diagram of the I²C unit.

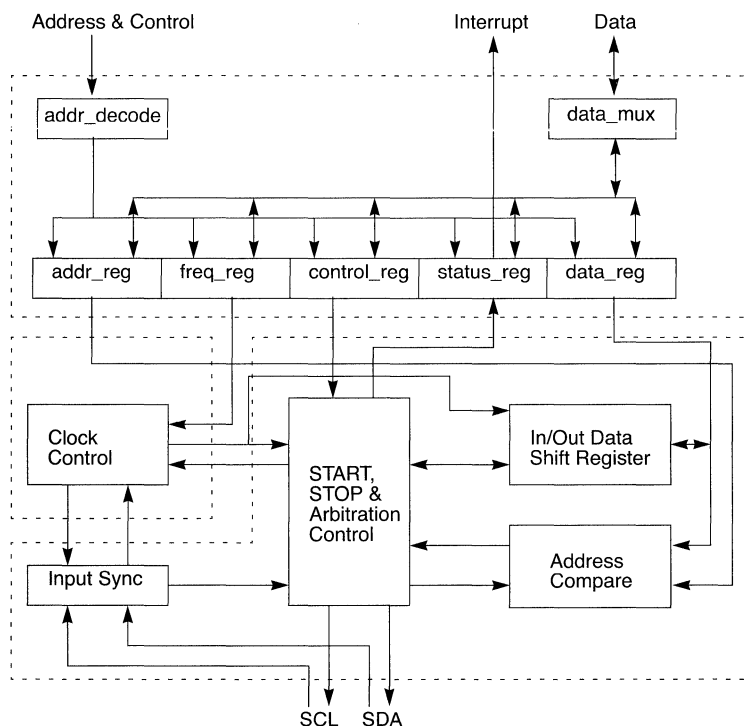


Figure 11-1. I²C Interface Block Diagram

11.2 I²C Protocol

A standard I²C transfer consists of four parts:

1. START condition
2. slave target address transmission
3. data transfer
4. STOP condition

Figure 11-2 shows the interaction of these four parts and the calling address, data byte, and new calling address components of the I²C protocol. The details of the protocol are described in the following subsection.

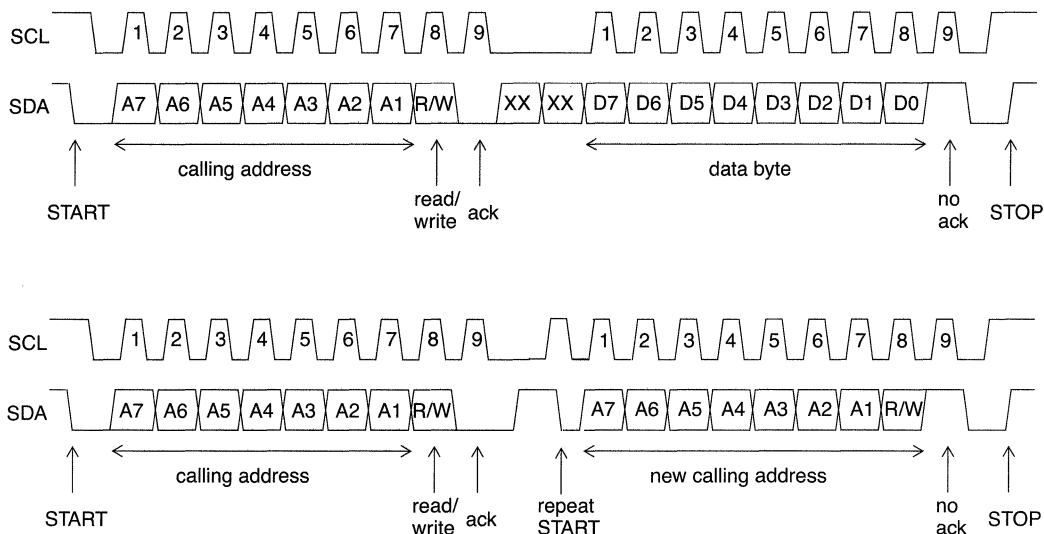


Figure 11-2. I²C Interface Transaction Protocol

11.2.1 START Condition

When no device is engaging the bus (both SDA and SCL lines are at logic high), a master can initiate a transfer by sending a START condition. As shown in Figure 11-2, a START condition is defined as a high-to-low transition of SDA while the SCL is high. This condition denotes the beginning of a new data transfer (each data transfer can contain several bytes of data) and awakens all slaves.

11.2.2 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/\bar{W} bit, which gives the slave the direction of the data being transferred. No two slaves in the system can have the same address. In addition, when the I²C device 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 (pulling the SDA signal low at the 9th clock) as shown in Figure 11-2. If no slave acknowledges the address, the master should generate a STOP condition or a repeated START condition.

11.2.3 Data Transfer

When successful slave addressing is achieved (and SCL returns to zero), the data transfer can proceed on a byte-to-byte basis in the direction specified by the R/\overline{W} bit sent by the calling master.

Each data byte is eight bits long. Data bits can be changed only while the SCL signal is low and must be held stable while the SCL signal is high, as shown in Figure 11-2. There is one clock pulse on SCL for each data bit, and the most significant bit (msb) is transmitted first. Each byte of data must be followed by an acknowledge bit, which is signalled from the receiving device by pulling the SDA line low at the 9th clock. Therefore, one complete data byte transfer takes nine clock pulses. Several bytes can be transferred during a data transfer session.

If the slave receiver does not acknowledge the master, the SDA line must be left high by the slave. The master can then generate a stop condition to abort the data transfer or a START condition (repeated START) to begin a new calling.

If the master receiver does not acknowledge the slave transmitter after a byte of transmission, it means to the slave that the “end of data” has been reached. Then the slave releases the SDA line for the master to generate a STOP or a START condition.

11.2.4 Repeated START Condition

As shown in Figure 11-2, a repeated START condition is a START condition generated without a STOP condition to terminate the previous transfer. The master uses this method to communicate with another slave or with the same slave in a different mode (transmit/receive mode) without releasing the bus.

11.2.5 STOP Condition

The master can terminate the transfer by generating a STOP condition to free the bus. A STOP condition is defined as a low-to-high transition of the SDA signal while SCL is high. For more information, see Figure 11-2. 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.

As described in Section 11.2.4, “Repeated START Condition,” the master can generate a START condition followed by a calling command without generating a STOP condition for the previous transfer. This is called a repeated START condition.

11.2.6 Arbitration Procedure

The I²C interface is a true multiple master bus that allows more than one master device to be connected on it. If two or more masters try to control the bus simultaneously, each master (including the MPC8240) has a clock synchronization procedure that 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’ while another master transmits a logic ‘0’. The losing masters immediately switch over to

slave-receive mode and stop driving the SDA line. In this case, the transition from master to slave mode does not generate a STOP condition. Meanwhile, the I²C unit sets the I2CSR[MAL] status bit to indicate the loss of arbitration.

Arbitration is lost (and I2CSR[MAL] is set) in the following circumstances:

- SDA sampled as low when the master drives a high during address or data-transmit cycle.
- SDA sampled as low when the master drives a high during the acknowledge bit of a data-receive cycle
- A START condition is attempted when the bus is busy.
- A repeated START condition is requested in slave mode.

Note that the MPC8240 does not automatically retry a failed transfer attempt.

If the I²C module of the MPC8240 is enabled in the middle of an ongoing byte transfer, the interface behaves as follows:

- Slave mode—the MPC8240 ignores the current transfer on the bus and starts operating whenever a subsequent START condition is detected.
- Master mode—the MPC8240 will not be aware that the bus is busy; therefore, if a START condition is initiated, the current bus cycle can become corrupt. This ultimately results in the current bus master of the I²C interface losing arbitration, after which bus operations return to normal.

11.2.7 Clock Synchronization

Due to the wire AND logic on the SCL line, a high-to-low transition on the SCL line affects all the devices connected on the bus. The devices begin counting their low period when the master drives the SCL line low. Once a device has driven SCL low, it holds the SCL line low until the clock high state is reached. However, the change of low-to-high in a device clock may not change the state of the SCL line if another device is still within its low period. Therefore, synchronized clock SCL is held low by the device with the longest low period. Devices with shorter low periods enter a high wait state during this time. When all devices concerned have counted off their low period, the synchronized SCL line is released and pulled high. There is then no difference between the devices' clocks and the state of the SCL line; and all the devices begin counting their high periods. The first device to complete its high period pulls the SCL line low again.

11.2.8 Handshaking

The clock synchronization mechanism can be used as a handshake in data transfer. Slave devices can hold the SCL low after completion of one byte transfer (9 bits). In such cases, it halts the bus clock and forces the master clock into wait states until the slave releases the SCL line.

11.2.9 Clock Stretching

Slaves can use the clock synchronization mechanism to slow down the transfer bit rate. After the master has driven the SCL line low, the slave can drive the SCL low for the required period and then release it. If the slave SCL low period is greater than the master SCL low period, then the resulting SCL bus signal low period is stretched.

11.3 Programming Model

There are five registers in the I²C unit that are used for the address, data, configuration, control, and status of the I²C interface. These registers are located in the embedded utilities memory block; see Section 4.4, “Embedded Utilities Memory Block (EUMB).” Table 11-2 summarizes the I²C registers.

Table 11-2. I²C Register Summary

Local Memory Offset	Register Name
0x3000	I ² C address register (I2CADR)
0x3004	I ² C frequency divider register (I2CFDR)
0x3008	I ² C control register (I2CCR)
0x300C	I ² C status register (I2CSR)
0x3010	I ² C data register (I2CDR)

NOTE

Even though reserved fields return 0, this should not be assumed by the programmer. Reserved bits should always be written with the value they returned when read. In other words, the register should be programmed by reading the value, modifying the appropriate fields, and writing back the value.

This does not apply to the I²C data register (I2CDR).

The I²C registers in this chapter are shown in little-endian format. If the system is big-endian, software must swap the bytes appropriately.

11.3.1 I²C Address Register (I2CADR)

The I2CADR, shown in Figure 11-3, contains the address that the I²C interface responds to when addressed as a slave. Note that it is not the address sent on the bus during the address calling cycle when the MPC8240 is in master mode.

 Reserved

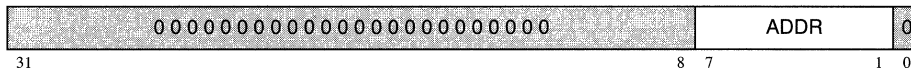


Figure 11-3. I²C Address Register (I2CADR)

Table 11-6 describes the bit settings of I2CADR.

Table 11-3. I2CADR Field Descriptions—Offset 0x0_3000

Bit	Name	Reset Value	R/W	Description
31–8	—	All zeros	R	Reserved
7–1	ADDR	0x00	R/W	Slave address. Contains the specific slave address 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.
0	—	0	R	Reserved

11.3.2 I²C Frequency Divider Register (I2CFDR)

The I2CFDR, shown in Figure 11-4, configures the sampling rate and the clock bit rate for the I²C unit.

 Reserved

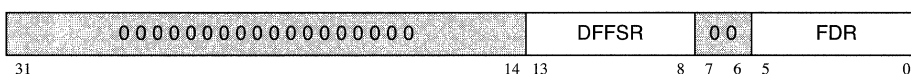


Figure 11-4. I²C Frequency Divider Register (I2CFDR)

Table 11-4 describes the bit settings of the I2CFDR.

Table 11-4. I2CFDR Field Descriptions—Offset 0x0_3004

Bit	Name	Reset Value	R/W	Description
31-14	—	All zeros	R	Reserved
13-8	DFFSR	0x10	R/W	Digital filter frequency sampling rate—To assist in filtering out signal noise, the sample rate is programmable; this field is used to prescale the frequency at which the digital filter takes samples from the I ² C bus. The resulting sampling rate is the local memory frequency (SDRAM_CLK) divided by the non-zero value set in this field. If DFFSR is set to zero, the I ² C bus sample points default to the reset divisor 0x10.
7-6	—	00	R	Reserved
5-0	FDR	0x00	R/W	Frequency divider ratio—Used to prescale the clock for bit rate selection. The serial bit clock frequency of SCL is equal to the local memory clock (SDRAM_CLK) divided by the divider shown in Table 11-5. Note that the frequency divider value can be changed at any point in a program.

Table 11-5 maps the I2CFDR[FDR] field to the clock divider values.

Table 11-5. Serial Bit Clock Frequency Divider Selections

FDR	Divider (Decimal)	FDR	Divider (Decimal)
0x00	288	0x20	160
0x01	320	0x21	192
0x02	384	0x22	224
0x03	480	0x23	256
0x04	576	0x24	320
0x05	640	0x25	384
0x06	768	0x26	448
0x07	960	0x27	512
0x08	1152	0x28	640
0x09	1280	0x29	768
0x0A	1536	0x2A	896
0x0B	1920	0x2B	1024
0x0C	2304	0x2C	1280
0x0D	2560	0x2D	1536
0x0E	3072	0x2E	1792

Table 11-5. Serial Bit Clock Frequency Divider Selections (Continued)

FDR	Divider (Decimal)	FDR	Divider (Decimal)
0x0F	3840	0x2F	2048
0x10	4608	0x30	2560
0x11	5120	0x31	3072
0x12	6144	0x32	3584
0x13	7680	0x33	4096
0x14	9216	0x34	5120
0x15	10240	0x35	6144
0x16	12288	0x36	7168
0x17	15360	0x37	8192
0x18	18432	0x38	10240
0x19	20480	0x39	12288
0x1A	24576	0x3A	14336
0x1B	30720	0x3B	16384
0x1C	36864	0x3C	20480
0x1D	40960	0x3D	24576
0x1E	49152	0x3E	28672
0x1F	61440	0x3F	32768

11.3.3 I²C Control Register (I2CCR)

The I2CCR controls the modes of the I²C interface, and is shown in Figure 11-5.

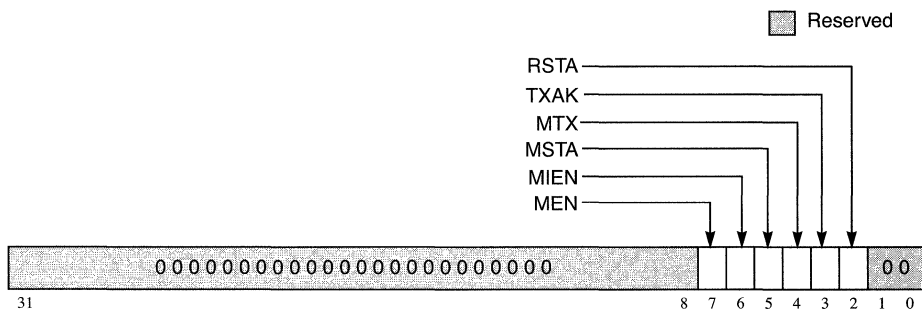


Figure 11-5. I²C Control Register (I2CCR)

Table 11-6 describes the bit settings of the I2CCR.

Table 11-6. I2CCR Field Descriptions—Offset 0x0_3008

Bit	Name	Reset Value	R/W	Description
31–8	—		R	Reserved
7	MEN	0	R/W	Module enable—This bit controls the software reset of the I ² C module. 0 The module is reset and disabled. When low, the interface is held in reset but the registers can still be accessed. 1 The I ² C module is enabled. This bit must be set before any other control register bits have any effect. All I ² C registers for slave receive or master START can be initialized before setting this bit. Refer to Section 11.2.6, “Arbitration Procedure.”
6	MIEN	0	R/W	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 I2CSR[MIF] is also set.
5	MSTA	0	R/W	Master/slave mode START 0 Slave mode—When this bit is changed from a 1 to 0, a STOP condition is generated and the mode changes from master to slave. 1 Master mode—When this bit is changed from a 0 to 1, a START condition is generated on the bus, and the master mode is selected. The MSTA bit is cleared without generating a STOP condition when the master loses arbitration. See Section 11.2.6, “Arbitration Procedure.”
4	MTX	0	R/W	Transmit/receive mode select—This bit selects the direction of the master and slave transfers. When configured as a slave, this bit should be set by software according to I2CSR[SRW]. In master mode, the bit should be set according to the type of transfer required. Therefore, for address cycles, this bit will always be high. 0 Receive mode 1 Transmit mode The MTX bit is cleared when the master loses arbitration.
3	TXAK	0	R/W	Transfer acknowledge—This bit specifies the value driven onto the SDA line during acknowledge cycles for both master and slave receivers. The value of this bit only applies when the I ² C module is configured as a receiver, not a transmitter. It also does not apply to address cycles; when the MPC8240 is addressed as a slave, an acknowledge is always sent. 0 An acknowledge signal (low value on SDA) is sent out to the bus at the 9th clock bit after receiving one byte of data. 1 No acknowledge signal response is sent (i.e., acknowledge value on SDA is high)
2	RSTA	0	W	Repeat START Setting this bit always generates a repeated START condition on the bus, provided the MPC8240 is the current bus master. Attempting a repeated START at the wrong time, (or if the bus is owned by another master), results in loss of arbitration. Note that this bit is not readable. 1 Generates repeat START condition
1–0	—	00	R	Reserved

11.3.4 I²C Status Register (I2CSR)

The status register, shown in Figure 11-6, is read-only with the exception of the MIF and MAL bits, which can be cleared by software. The MCF and RXAK bits are set at reset; all other I2CSR bits are cleared on reset.

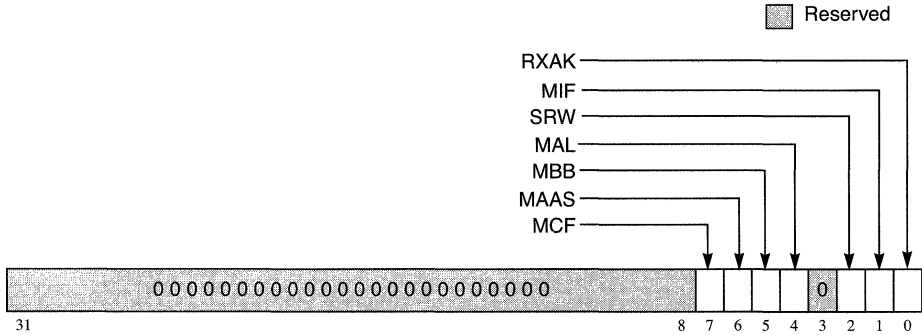


Figure 11-6. I²C Status Register (I2CSR)

Table 11-7 describes the bits settings of the I2CSR.

Table 11-7 I2CSR Field Descriptions—Offset 0x0_300C

Bit	Name	Reset Value	R/W	Description
31–8	—	0	R	Reserved
7	MCF	1	R	Data transferring—While one byte of data is being transferred, this bit is cleared. It is set by the falling edge of the 9th clock of a byte transfer. 0 Transfer in progress. MCF is cleared when I2C DR is read in receive mode or when I2C DR is written in transmit mode. 1 Transfer complete
6	MAAS	0	R	Addressed as a slave—When the value in I2C ADR matches with the calling address, this bit is set. The processor is interrupted, provided I2C CR[M IEN] is set. Next, the processor must check the SRW bit and set I2C CR[M TX] accordingly. Writing to the I2C CR automatically clears this bit. 0 Not addressed as a slave 1 Addressed as a slave
5	MBB	0	R	Bus busy—This bit 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
4	MAL	0	R/W	Arbitration lost—This bit is automatically set when the arbitration procedure is lost. Note that the MPC8240 does not automatically retry a failed transfer attempt. 0 Arbitration is not lost. Can only be cleared by software. 1 Arbitration is lost. See Section 11.2.6, “Arbitration Procedure.”

Table 11-7 I2CSR Field Descriptions—Offset 0x0_300C (Continued)

Bit	Name	Reset Value	R/W	Description
3	—	0	R	Reserved
2	SRW	0	R	<p>Slave read/write—When MAAS is set, SRW indicates the value of the R/W command bit of the calling address sent from the master.</p> <p>0 Slave receive, master writing to slave 1 Slave transmit, master reading from slave</p> <p>This bit is valid only when:</p> <ul style="list-style-type: none"> • a complete transfer has occurred and no other transfers have been initiated, and • the I²C interface is configured as a slave and has an address match. <p>By checking this bit, the processor can select slave transmit/receive mode according to the command of the master.</p>
1	MIF	0	R/W	<p>Module interrupt—The MIF bit is set when an interrupt is pending, causing a processor interrupt request (provided I2CCR[MIEN] is set).</p> <p>0 No interrupt pending. Can only be cleared by software 1 Interrupt pending. MIF is set when one of the following events occur:</p> <ul style="list-style-type: none"> • One byte of data is transferred (set at the falling edge of the 9th clock), • The value in I2CADR matches with the calling address in slave-receive mode, or • Arbitration is lost. See Section 11.2.6, “Arbitration Procedure.”
0	RXAK	1	R	<p>Received acknowledge—The value of SDA during the 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.</p> <p>0 Acknowledge received 1 No acknowledge received</p>

11.3.5 I²C Data Register (I2CDR)

The data register is shown in Figure 11-7.

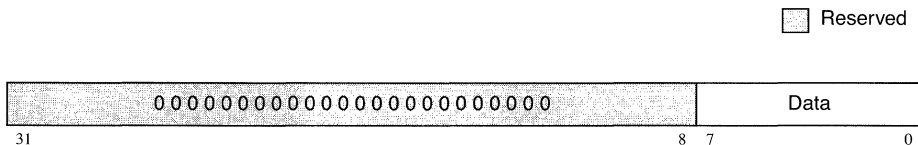


Figure 11-7. I²C Data Register (I2CDR)

Table 11-8 describes I2CDR.

Table 11-8. I2CDR Field Descriptions—Offset 0x0_3010

Bit	Name	Reset Value	R/W	Description
31–8	—		R	Reserved
7–0	Data	0x00	R/W	Transmission starts when an address and the R/W bit are written to the data register and the I ² C interface is the master. A data transfer is initiated when data is written to the I2CDR. The most significant bit (msb) is sent first in both cases. In the master receive mode, reading the data register allows the read to occur but also initiates next byte data receiving. In slave mode, the same function is available after it is addressed.

11.4 Programming Guidelines

This section describes some programming guidelines recommended for the I²C interface on the MPC8240. Also included is a recommended flowchart for the I²C interrupt service routines.

The I²C registers in this chapter are shown in little-endian format. If the system is in big-endian mode, software must swap the bytes appropriately. Also, a SYNC assembler instruction should be executed after each I²C register read/write access to guarantee in-order execution.

The MPC8240 does not guarantee it will recover 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 status bits returned after an interrupt are not consistent with what was expected due to illegal I²C bus protocol behavior.

Example I²C code can be found in the MPC8240 Device Driver Toolbox available through the PowerPC web site www.mot.com/SPS/PowerPC/teksupport/faqsolutions/code.

11.4.1 Initialization Sequence

A hard reset initializes all the I²C registers to their default states. The following initialization sequence must be used before using the I²C unit:

1. If the processor’s memory management unit (MMU) is enabled, all I²C registers must be located in a cache-inhibited area.
2. Program the embedded utilities memory block; see Section 4.4, “Embedded Utilities Memory Block (EUMB).” Valid locations are limited from 0x800 to 0xFDF.
3. Update I2CFDR[FDR] and select the required division ratio to obtain the SCL frequency from the local memory clock (SDRAM_CLK).

4. Update the I2CADR to define the slave address for this device.
5. Modify I2CCR to select master/slave mode, transmit/receive mode, and interrupt-enable or disable.
6. Set the I2CCR[MEN] to enable the I²C interface.

11.4.2 Generation of START

After initialization, the following sequence can be used to generate START:

1. Select master mode (set I2CCR[MSTA]) to transmit serial data. If the MPC8240 is connected to a multi-master I²C system, test the state of I2CSR[MBB] to check whether the serial bus is free (I2CSR[MBB] = 0) before switching to master mode.
2. Set I2CCR[MSTA].
3. Write the slave address being called into the data register (I2CDR). The data written to I2CDR comprises the slave-calling address; and the least significant bit (lsb) is set to indicate the direction of transfer (transmit/receive) required from the slave.
4. Set I2CCR[MTX] for the address cycle.

The above scenario assumes the I²C interrupt bit (I2CSR[MIF]) is cleared. If I2CSR[MIF] = 1 at any time, the I²C interrupt handler should immediately handle the interrupt. See Section 11.4.7, “Interrupt Service Routine Flowchart.”

11.4.3 Post-Transfer Software Response

Transmission or reception of a byte automatically sets the data transferring bit (I2CSR[MCF]), which indicates that one byte has been transferred. The I²C interrupt bit (I2CSR[MIF]) is also set; an interrupt is generated to the processor if the interrupt function is enabled during the initialization sequence by setting I2CCR[MIEN]. In the interrupt handler,

- Software must clear I2CSR[MIF].
- Software must read the contents of the I²C data register (I2CDR) in receive mode or write to I2CDR in transmit mode. Note that this causes I2CSR[MCF] to be cleared. See Section 11.4.7, “Interrupt Service Routine Flowchart.”

When an interrupt occurs at the end of the address cycle, the master remains in transmit mode. If master receive mode is required, indicated by the R \overline{W} bit (SRW) in I2CSR, I2CCR[MTX] should be toggled at this stage. See Section 11.4.7, “Interrupt Service Routine Flowchart.”

Software can service the I2CDR in the main program by monitoring I2CSR[MIF] if the interrupt function is disabled. In this case, I2CSR[MIF] should be polled, rather than I2CSR[MCF] because MCF behaves differently when arbitration is lost.

During slave-mode address cycles (I2CSR[MAAS] = 1), I2CSR[SRW] should be read to determine the direction of the subsequent transfer and I2CCR[MTX] should be

programmed accordingly. For slave-mode data cycles ($I2CSR[MAAS] = 0$), $I2CSR[SRW]$ is not valid. $I2CCR[MTX]$ should be read to determine the direction of the current transfer. See Section 11.4.7, “Interrupt Service Routine Flowchart,” for more details.

11.4.4 Generation of STOP

A data transfer ends with a STOP condition generated by the master device. A master transmitter can generate a STOP condition after all the data has been transmitted.

If a master receiver wants to terminate a data transfer, it must inform the slave transmitter by not acknowledging the last byte of data, which is done by setting the transmit acknowledge ($I2CCR[TXAK]$) bit before reading the next-to-last byte of data. For one-byte transfers, a dummy read should be performed by the interrupt service routine. (See Section 11.4.7, “Interrupt Service Routine Flowchart.”) Before the interrupt service routine reads the last byte of data, a STOP condition must first be generated by the MPC8240. The MPC8240 automatically generates a STOP if $I2CCR[TXAK] = 1$.

11.4.5 Generation of Repeated START

At the end of a data transfer, if the master still wants to communicate on the bus, it can generate another START condition followed by another slave address without first generating a STOP condition by setting $I2CCR[RSTA]$.

11.4.6 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 just been received. If $I2CSR[MAAS] = 1$, software should set the transmit/receive mode select bit ($I2CCR[MTX]$) according to the R/W command bit ($I2CSR[SRW]$). Writing to $I2CCR$ clears $I2CSR[MAAS]$ automatically. The only time $I2CSR[MAAS]$ is read as set is from the interrupt handler at the end of that address cycle where an address match occurred; interrupts resulting from subsequent data transfers will have $I2CSR[MAAS] = 0$. A data transfer can then be initiated by writing to $I2CDR$ for slave transmits or dummy reading from $I2CDR$ in slave-receive mode. The slave drives SCL low between byte transfers. SCL is released when the $I2CDR$ is accessed in the required mode.

11.4.6.1 Slave Transmitter and Received Acknowledge

In the slave transmitter routine, the received acknowledge bit ($I2CSR[RXAK]$) must be tested before sending the next byte of data. The master signals an end-of-data by not acknowledging the data transfer from the slave. When no acknowledge is received ($I2CSR[RXAK] = 1$), the slave transmitter interrupt routine must clear $I2CCR[MTX]$ to switch the slave from transmitter to receiver mode. A dummy read of $I2CDR$ then releases SCL so that the master can generate a STOP condition. See also Section 11.4.7, “Interrupt Service Routine Flowchart.”

11.4.6.2 Loss of Arbitration and Forcing of Slave Mode

When a master loses arbitration (see Section 11.2.6, “Arbitration Procedure”), I2CSR[MAL] is set (indicating loss of arbitration); I2CCR[MSTA] is cleared (changing the master to slave mode), and an interrupt occurs (if enabled) at the falling edge of the 9th clock of this transfer. Thus, the slave interrupt service routine should test I2CSR[MAL] first and the software should clear I2CSR[MAL] if it is set.

11.4.7 Interrupt Service Routine Flowchart

Figure 11-8 shows an example algorithm for an I²C interrupt service routine. Deviation from the flowchart may result in unpredictable I²C bus behavior.

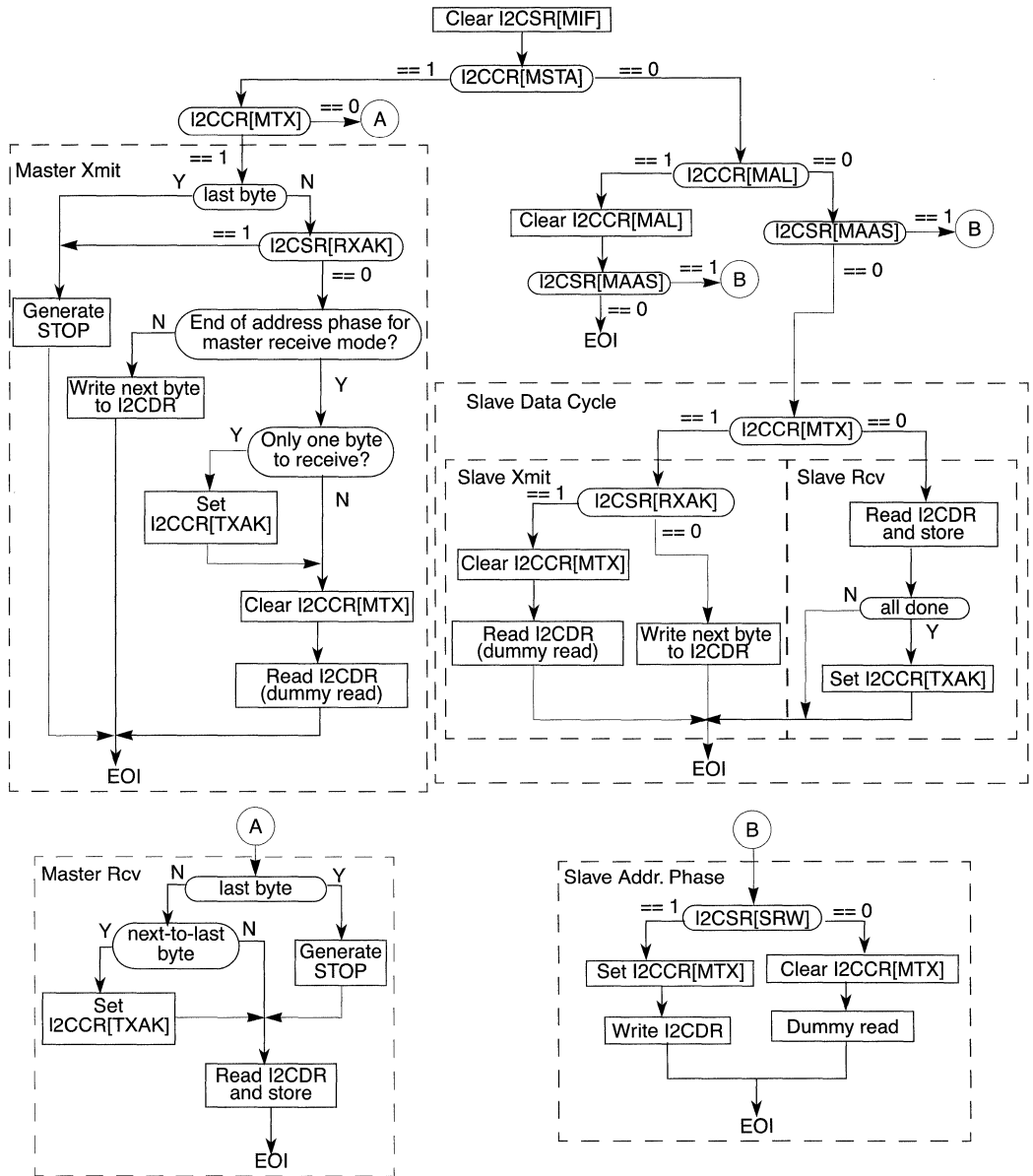


Figure 11-8. Example I²C Interrupt Service Routine Flowchart

Chapter 12

Embedded Programmable Interrupt Controller (EPIC)

This chapter describes the embedded programmable interrupt controller (EPIC) interrupt protocol, the various types of interrupt sources controlled by the EPIC unit and a complete description of the EPIC registers with some programming guidelines.

The interrupt sources and soft reset controlled by the EPIC unit, the external $\overline{\text{SRESET}}$ signal, and the internal $\overline{\text{mcp}}$ generated by the MPC8240 central control unit (CCU) all cause exceptions in the processor core. The $\overline{\text{int}}$ signal is the main interrupt output from the EPIC to the processor core and causes the external interrupt exception. The external $\overline{\text{SRESET}}$ signal or the internal $\overline{\text{sreset}}$ output of the EPIC unit cause the soft reset processor exception. The machine check exception is caused by the internal $\overline{\text{mcp}}$ signal generated by the CCU, informing the processor of error conditions, the assertion of the external NMI signal, and other conditions. See Chapter 13, “Error Handling and Exceptions,” for further information on the sources of the $\overline{\text{mcp}}$ internal signal.

12.1 EPIC Unit Overview

The (EPIC) implements the necessary functions to provide a flexible and general-purpose interrupt controller solution. The EPIC pools hardware-generated interrupts from many sources, both within the MPC8240 and externally, and delivers them to the processor core in a prioritized manner. The solution adopts the OpenPIC architecture (architecture developed jointly by AMD and Cyrix for SMP interrupt solutions) and implements the logic and programming structures according to that specification. The MPC8240’s EPIC unit supports up to five external interrupts, four internal logic-driven interrupts, four timers with interrupts, one serial-style interrupt line (supporting 16 interrupts) and a pass-through mode.

The following sections give an overview of the features of the EPIC unit and a complete summary of the signals used by the EPIC unit.

12.1.1 EPIC Features Summary

The EPIC unit of the MPC8240 implements the following features:

- OpenPIC programming model
- Support for five external interrupt sources or one serial-style interrupt (16 interrupt sources)
- Four global high-resolution timers that can be interrupt sources
- Interrupt control for the MPC8240 I²C, DMA (2 channels) and I₂O units
- Support for connection of external interrupt controller device such as an 8259 Programmable Interrupt Controller (PIC)
- In 8259-mode, it generates local (internal) interrupts output signal, $\overline{L_INT}$.
- Processor initialization control—The processor can reset itself by setting the processor initialization register, causing the assertion of the internal *sreset* signal as described in Section 12.9.5, “Processor Initialization Register (PI).”
- Programmable resetting of the EPIC unit through the global configuration register
- 16 programmable interrupt priority levels
- Fully-nested interrupt delivery
- Spurious vector generation
- 32-bit configuration registers that are aligned on 128-bit boundaries

12.1.2 EPIC Interface Signal Description

The external EPIC signals are defined in Table 12-1.

Table 12-1. EPIC Interface Signal Description

Signal Name	Pins	I/O	State Meaning
IRQ0/S_INT	1	I	Direct IRQ mode—input representing an incoming interrupt request. Serial IRQ mode: input representing the serial interrupt data stream. Note: IRQ0 is used when operating in the pass-through mode.
IRQ1/S_CLK	1	I/O	Direct IRQ mode—input representing an incoming interrupt request Serial IRQ mode: output representing the serial clock by which the remote sequencer (interrupt source) clocks serial interrupts out.
IRQ2/S_RST	1	I/O	Direct IRQ mode—input representing an incoming interrupt request Serial IRQ mode: output pulse is active after EPIC resets and is set to serial mode. It determines the serial interrupt slot count for all external serial devices. Refer to Figure 12-3.
IRQ3/ $\overline{S_FRAME}$	1	I/O	Direct IRQ mode—input representing an incoming interrupt request Serial IRQ mode: output that pulses low each time the interrupt controller is sampling interrupt source 0.
IRQ4/ $\overline{L_INT}$	1	I/O	Direct IRQ mode—input representing an incoming interrupt request Serial IRQ mode: not used. Pass-through mode: output active low whenever there is an interrupt from MPC8240's internal dma0, dma1, I ₂ O(message unit), or I ² C units.

12.1.3 EPIC Block Diagram

The EPIC unit in the MPC8240 is accessible from the processor only. The processor reads and writes the configuration and status registers of EPIC. These registers are memory mapped.

The following block diagram shows the EPIC unit and its relationship to the processor core and external signals:

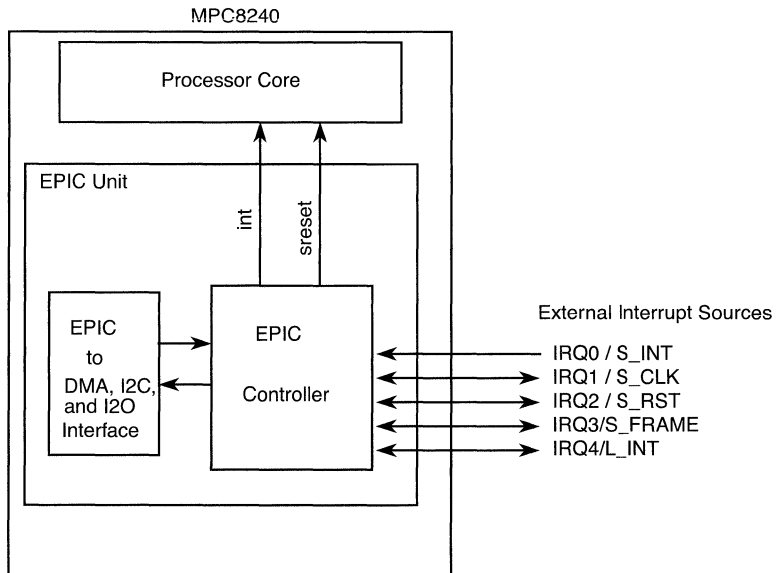


Figure 12-1. EPIC Unit Block Diagram

12.2 EPIC Register Summary

The EPIC register map occupies a 256-Kbyte range of the embedded utilities memory block (EUMB). For further details, see Section 4.4, “Embedded Utilities Memory Block (EUMB).” If an access is attempted to an undefined portion of the map, the device returns 0x0000_0000 as its value on reads and does nothing on writes.

All EPIC registers are 32 bits wide and reside on 128-bit address boundaries. All addresses mentioned in this chapter are offsets from the EUMBBAR located at 0x78; see Section 5.5, “Embedded Utilities Memory Block Base Address Register.”

The EPIC address offset map is divided into the following four distinct areas:

- 0xnn4_1000 - 0xnn4_10F0—Global EPIC register map
- 0xnn4_1100 - 0xnn4_FFF0—Global timer register map
- 0xnn5_0000 - 0xnn5_FFF0—Interrupt source configuration register map
- 0xnn6_0000 - 0xnn6_0FF0—Processor-related register map

Table 12-2 defines the address map for the global EPIC and timer registers.

Table 12-2. EPIC Register Address Map—Global and Timer Registers

Address Offset from EUMBBAR	Register Name	Field Mnemonics
0x4_1000	Feature reporting register (FRR)	NIRQ, NCPU, VID
0x4_1010	Reserved	—
0x4_1020	Global configuration register (GCR)	R (reset), M (mode)
0x4_1030	EPIC interrupt configuration register (EICR)	R (clock ratio), SIE
0x4_1040–0x4_1070	Reserved	—
0x4_1080	EPIC vendor identification register (EVI)	STEP, DEVICE_ID, VENDOR_ID
0x4_1090	Processor initialization register (PI)	P0
0x4_10A0–0x4_10D0	Reserved	—
0x4_10E0	Spurious vector register (SVR)	VECTOR
0x4_10F0	Timer frequency reporting register (TFRR)	TIMER_FREQ
0x4_1100	Global timer 0 current count register (GTCCR0)	T (toggle), COUNT
0x4_1110	Global timer 0 base count register (GTBCR0)	CI, BASE_COUNT
0x4_1120	Global timer 0 vector/priority register (GTVPR0)	M, A, PRIORITY, VECTOR
0x4_1130	Global timer 0 destination register (GTDR0)	P0
0x4_1140	Global timer 1 current count register (GTCCR1)	T (toggle), COUNT
0x4_1150	Global timer 1 base count register (GTBCR1)	CI, BASE_COUNT
0x4_1160	Global timer 1 vector/priority register (GTVPR1)	M, A, PRIORITY, VECTOR
0x4_1170	Global timer 1 destination register (GTDR1)	P0
0x4_1180	Global timer 2 current count register (GTCCR2)	T (toggle), COUNT
0x4_1190	Global timer 2 base count register (GTBCR2)	CI, BASE_COUNT
0x4_11A0	Global timer 2 vector/priority register (GTVPR2)	M, A, PRIORITY, VECTOR
0x4_11B0	Global timer 2 destination register (GTDR2)	P0
0x4_11C0	Global timer 3 current count register (GTCCR3)	T (toggle), COUNT
0x4_11D0	Global timer 3 base count register (GTBCR3)	CI, BASE_COUNT
0x4_11E0	Global timer 3 vector/priority register (GTVPR3)	M, A, PRIORITY, VECTOR
0x4_11F0	Global timer 3 destination register (GTDR3)	P0
0x4_1200–0x5_01F0	Reserved	—

Table 12-3 defines the address map for the interrupt source configuration registers.

Table 12-3. EPIC Register Address Map—Interrupt Source Configuration Registers

Address Offset from EUMBBAR	Register Name	Field Mnemonics
0x5_0200	IRQ0 vector/priority register (IVPR0)	M, A, P, S, PRIORITY, VECTOR
0x5_0210	IRQ0 destination register (IDR0)	P0
0x5_0220	IRQ1 vector/priority register (IVPR1)	M, A, P, S, PRIORITY, VECTOR
0x5_0230	IRQ1 destination (IDR1)	P0
0x5_0240	IRQ2 vector/priority register (IVPR2)	M, A, P, S, PRIORITY, VECTOR
0x5_0250	IRQ2 destination (IDR2)	P0
0x5_0260	IRQ3 vector/priority register (IVPR3)	M, A, P, S, PRIORITY, VECTOR
0x5_0270	IRQ3 destination (IDR3)	P0
0x5_0280	IRQ4 vector/priority register (IVPR4)	M, A, P, S, PRIORITY, VECTOR
0x5_0290	IRQ4 destination (IDR4)	P0
0x5_00A0–0x5_01F0	Reserved	—
0x5_0200	Serial interrupt 0 vector/priority register (SVPR0)	M, A, P, S, PRIORITY, VECTOR
0x5_0210	Serial interrupt 0 destination register (SDR0)	P0
0x5_0220	Serial interrupt 1 vector/priority register (SVPR1)	M, A, P, S, PRIORITY, VECTOR
0x5_0230	Serial interrupt 1 destination register (SDR1)	P0
0x5_0240	Serial interrupt 2 vector/priority register (SVPR2)	M, A, P, S, PRIORITY, VECTOR
0x5_0250	Serial interrupt 2 destination register (SDR2)	P0
0x5_0260	Serial interrupt 3 vector/priority register (SVPR3)	M, A, P, S, PRIORITY, VECTOR
0x5_0270	Serial interrupt 3 destination register (SDR3)	P0
0x5_0280	Serial interrupt 4 vector/priority register (SVPR4)	M, A, P, S, PRIORITY, VECTOR
0x5_0290	Serial interrupt 4 destination register (SDR4)	P0
0x5_02A0	Serial interrupt 5 vector/priority register (SVPR5)	M, A, P, S, PRIORITY, VECTOR
0x5_02B0	Serial interrupt 5 destination register (SDR5)	P0
0x5_02C0	Serial interrupt 6 vector/priority register (SVPR6)	M, A, P, S, PRIORITY, VECTOR
0x5_02D0	Serial interrupt 6 destination register (SDR6)	P0
0x5_02E0	Serial interrupt 7 vector/priority register (SVPR7)	M, A, P, S, PRIORITY, VECTOR
0x5_02F0	Serial interrupt 7 destination register (SDR7)	P0
0x5_0300	Serial interrupt 8 vector/priority register (SVPR8)	M, A, P, S, PRIORITY, VECTOR
0x5_0310	Serial interrupt 8 destination register (SDR8)	P0
0x5_0320	Serial interrupt 9 vector/priority register (SVPR9)	M, A, P, S, PRIORITY, VECTOR

Table 12-3. EPIC Register Address Map—Interrupt Source Configuration Registers

Address Offset from EUMBBAR	Register Name	Field Mnemonics
0x5_0330	Serial interrupt 9 destination register (SDR9)	P0
0x5_0340	Serial interrupt 10 vector/priority register (SVPR10)	M, A, P, S, PRIORITY, VECTOR
0x5_0350	Serial interrupt 10 destination register (SDR10)	P0
0x5_0360	Serial interrupt 11 vector/priority register (SVPR11)	M, A, P, S, PRIORITY, VECTOR
0x5_0370	Serial interrupt 11 destination register (SDR11)	P0
0x5_0380	Serial interrupt 12 vector/priority register (SVPR12)	M, A, P, S, PRIORITY, VECTOR
0x5_0390	Serial interrupt 12 destination register (SDR12)	P0
0x5_03A0	Serial interrupt 13 vector/priority register (SVPR13)	M, A, P, S, PRIORITY, VECTOR
0x5_03B0	Serial interrupt 13 destination register (SDR13)	P0
0x5_03C0	Serial interrupt 14 vector/priority register (SVPR14)	M, A, P, S, PRIORITY, VECTOR
0x5_03D0	Serial interrupt 14 destination register (SDR14)	P0
0x5_03E0	Serial interrupt 15 vector/priority register (SVPR15)	M, A, P, S, PRIORITY, VECTOR
0x5_03F0	Serial interrupt 15 destination register (SDR15)	P0
0x5_0400–0x5_1010	Reserved	—
0x5_1020	I ² C interrupt vector/priority register (IIVPR0)	M, A, PRIORITY, VECTOR
0x5_1030	I ² C interrupt destination register (IIDR0)	P0
0x5_1040	DMA Ch0 interrupt vector/priority register (IIVPR1)	M, A, PRIORITY, VECTOR
0x5_1050	DMA Ch0 interrupt destination register (IIDR1)	P0
0x5_1060	DMA Ch1 interrupt vector/priority register (IIVPR2)	M, A, PRIORITY, VECTOR
0x5_1070	DMA Ch1 interrupt destination register (IIDR2)	P0
0x5_1080–0x5_10B0	Reserved	—
0x5_10C0	Message unit interrupt vector/priority register (IIVPR3)	M, A, PRIORITY, VECTOR
0x5_10D0	Message unit interrupt destination register (IIDR3)	P0
0x5_10E0–0x5_FFF0	Reserved	—

Table 12-4 defines the address map for the processor-related registers.

Table 12-4. EPIC Register Address Map—Processor-Related Registers

Address Offset from EUMBBAR	Register Name	Field Mnemonics
0x6_0000–0x6_0070	Reserved	—
0x6_0080	Processor current task priority register (PCTPR)	TASKP
0x6_0090	Reserved	—

Table 12-4. EPIC Register Address Map—Processor-Related Registers (Continued)

Address Offset from EUMBBAR	Register Name	Field Mnemonics
0x6_00A0	Processor interrupt acknowledge register (IACK)	VECTOR
0x6_00B0	Processor end-of-interrupt register (EOI)	EOI CODE
0x6_00C0–0x6_3FF0	Reserved	—

12.3 EPIC Unit Interrupt Protocol

The following sections describe the priority of interrupts controlled by the EPIC unit, the interrupt acknowledge mechanism, the nesting of multiple interrupts, and the handling of spurious interrupts.

12.3.1 Interrupt Source Priority

The software assigns a priority value to each interrupt source by writing to the vector/priority register for the particular source. Priority values are in the range 0 to 15 of which 15 is the highest. In order for an interrupt to be signalled to the processor, the priority of the source must be greater than that of the current task priority of the processor (and the in-service interrupt source priority). Therefore, setting a source priority to zero inhibits that interrupt.

12.3.2 Processor Current Task Priority

The processor has a task priority register (PCTPR) set by system software to indicate the relative importance of the task running on that processor. When an interrupt has a priority level greater than the current task priority (and the in-service interrupt source priority), it is signalled to the processor. Therefore, setting the current task priority to 15 for the processor prevents the signalling of any interrupt to the processor.

12.3.3 Interrupt Acknowledge

The EPIC unit notifies the processor core of an interrupt by asserting the \overline{int} signal. When the processor acknowledges the interrupt request by reading the interrupt acknowledge register (IACK) in the EPIC unit, the EPIC returns the 8-bit interrupt vector associated with the interrupt source to the processor via the internal data bus. The interrupt is then considered to be in service, and it remains in service until the processor performs a write to the EPIC unit end of interrupt (EOI) register. Writing to the EOI register is referred to as an EOI cycle.

12.3.4 Nesting of Interrupts

If the processor core is servicing an interrupt, it can only be interrupted again if the EPIC unit receives an interrupt request from an interrupt source with a greater priority than the one currently being serviced.

Thus, although several interrupts may be simultaneously in service in the processor, the code currently executing is always for handling the highest priority of all that are in service. When the processor performs an EOI cycle, this highest priority interrupt is taken out of service. The next EOI cycle takes the next highest priority interrupt out of service, and so forth.

12.3.5 Spurious Vector Generation

Under certain circumstances, the EPIC may not have a valid vector to return to the processor during an interrupt acknowledge cycle (for example, if there is not a pending interrupt with a sufficient priority level). In these cases, the spurious vector from the spurious vector register is returned. The following cases cause a spurious vector fetch:

- \overline{int} is asserted in response to an externally sourced interrupt which is activated with level sensitive logic, and the asserted level is negated before the interrupt is acknowledged.
- \overline{int} is asserted for an interrupt source that is later masked using the mask bit in the vector/priority register before the interrupt is acknowledged.
- \overline{int} is asserted for an interrupt source that is later masked by an increase in the task priority level before the interrupt is acknowledged.
- \overline{int} is not asserted (that is, software performs an IACK when there was no interrupt condition).
- \overline{int} is asserted when there is an illegal clock ratio value in the EPIC interrupt configuration register.

12.3.6 Internal Block Diagram Description

The internal block diagram shown in Figure 12-2 shows the interaction of the non-programmable EPIC registers and the interrupt delivery logic (assertion of the \overline{int} signal to the processor).

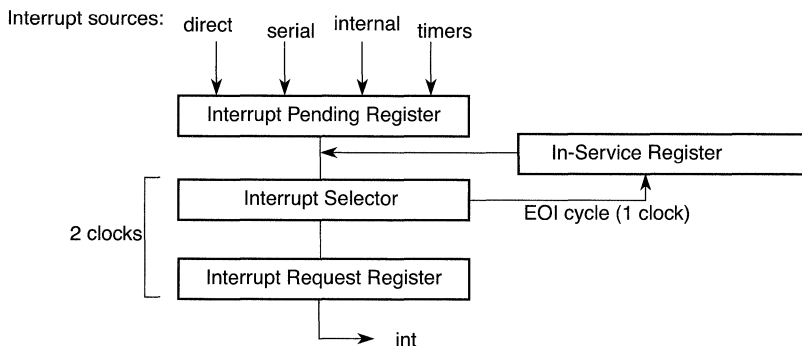


Figure 12-2. EPIC Interrupt Generation Block Diagram—Non-programmable Registers

12.3.6.1 Interrupt Pending Register (IPR)—Non-programmable

The interrupt signals in the EPIC unit are qualified and synchronized by the internal IPR, which has one bit for each interrupt. The mask bits from the appropriate vector/priority register are used to qualify the output of the IPR. Therefore, if an interrupt condition is detected when the mask bit is set, that interrupt is requested when the mask bit is cleared.

The interrupt source are internal (I²C, DMA or I₂O), EPIC (four timers), and external (interrupt signal [0-4] or 16 serial interrupts). When there is a direct or serial interrupt and the sense bit = 0 (edge-sensitive), the IPR is cleared when the interrupt associated with a particular bit in the IPR register is acknowledged with an interrupt acknowledge cycle. When the sense bit = 1 (level-activated), the IPR is not cleared until the source signal is negated.

Since an edge-sensitive interrupt is not cleared until it is acknowledged and the default polarity/sense bits for all interrupts are set to edge-sensitive at power-up, it is possible for EPIC to store detections of edges as pending interrupts. If software permanently sets the polarity/sense of an interrupt source to edge-sensitive and clears its mask bit, it can receive the vector for the interrupt source and not a spurious interrupt. To prevent having to handle a false interrupt, see the programming note in Section 12.8, “Programming Guidelines.”

12.3.6.2 Interrupt Selector (IS)

The interrupt selector (IS) receives interrupt requests from the IPR. The output of the IS is the highest priority interrupt that has been qualified. This output is the priority of the selected interrupt and its source identification. The IS resolves an interrupt request in two clocks.

During an EOI cycle, the value in the in-service register (ISR) is used to select which bits are to be cleared in the ISR. One cycle after an EOI cycle, the output of the IS is the interrupt source identification and priority value to be cleared from the ISR. This interrupt source is the one with highest priority in the in-service register.

12.3.6.3 Interrupt Request Register (IRR)

The interrupt request register (IRR) always passes the output of the IS except during interrupt acknowledge cycles. This guarantees that the vector that is read from the interrupt acknowledge register is not changing due to the arrival of a higher priority interrupt. The IRR also serves as a pipeline register for the two-clock propagation time through the IS.

12.3.6.4 In-Service Register (ISR)

The contents of the in-service register (ISR) are the priority and source values of the interrupts that are currently in service in the processor. The ISR receives an internal bit-set command during interrupt acknowledge cycles and an internal bit-clear command during EOI cycles.

12.4 EPIC Pass-Through Mode

The EPIC unit provides a mechanism to support alternate external interrupt controllers such as the 8259 interrupt controller architecture, PC-AT-compatible. After a hard reset, the EPIC unit defaults to pass-through mode. In this mode, interrupts from external source IRQ0 are passed directly to the processor; thus, the interrupt signal from the external interrupt controller can be connected to IRQ0, and cause direct interrupts to the processor. Note that IRQ0/S_INT is an active-high signal. However, note that the EPIC unit does not perform a vector fetch from an 8259 interrupt controller.

When pass-through mode is enabled, none of the internally generated interrupts are forwarded to the processor. However, in pass-through mode, the EPIC unit passes the raw interrupts from the I₂O, I²C, and DMA controllers to the $\overline{L_INT}$ output signal.

If the interrupts from the global timers, the message unit (I₂O), I²C, or DMA controllers are required to be reported internally to the processor, pass-through mode must be disabled. If pass-through mode is disabled, the internal and external interrupts are delivered using the priority and delivery mechanisms otherwise defined for the EPIC unit.

The pass-through mode is controlled by the GCR[M] bit. Note that when switching the EPIC unit from pass-through to mixed mode (either direct or serial), the programming note in Section 12.8, “Programming Guidelines,” may apply.

12.5 EPIC Direct Interrupt Mode

In direct interrupt mode, the IRQ[0–4] signals represent external interrupts that are controlled and prioritized by the five IRQ vector/priority registers (IVPR0–4), and the five IRQ destination registers (IDR0–4). The external interrupts can be programmed for either level- or edge-sensitive activation and either polarity. The direct interrupt mode is selected when EICR[SIE] = 0. Note that EICR[SIE] only has meaning when GCR[M] = 1 (direct mode is a subset of mixed-mode operation).

12.6 EPIC Serial Interrupt Interface

The serial interrupt mode is selected when EICR[SIE] = 1. Note that EICR[SIE] only has meaning when GCR[M] = 1 (serial interrupt mode is also a subset of mixed-mode operation). When the MPC8240 is in serial interrupt mode, 16 interrupt sources are supported through the following serial interrupt signals (that are multiplexed with the IRQ[0–3] input signals used in direct interrupt mode):

- Serial interrupt (S_INT) input signal
- Serial clock (S_CLK) output signal
- Serial reset (S_RST) output signal
- Serial frame ($\overline{S_FRAME}$) output signal

The 16 serial interrupts are controlled and prioritized by the 16 serial vector/priority registers (SVPR0–15) and the 16 serial destination registers (SDR0–15).

12.6.1 Sampling of Serial Interrupts

When the EPIC unit is programmed for serial interrupts, 16 sources are sampled through the S_INT input signal. Each source (0–15) is allocated a one-cycle time slot in a sequence of 16 cycles in which to request an interrupt. The serial interrupt interface is clocked by the EPIC S_CLK output. This clock can be programmed to run at 1/2 to 1/14 of the MPC8240's SDRAM_CLK frequency by appropriately setting a 3-bit field in the serial interrupt configuration register. See Section 12.9.3, “EPIC Interrupt Configuration Register (EICR).” Extreme care should be used with regard to board noise problems if a frequency above 33 MHz is chosen. All references to the clock and to cycles in this subsection refer to the S_CLK clock.

When EPIC is switched to serial mode by setting EICR[SIE] = 1, a 16-cycle sequence begins 4 S_CLK cycles after EPIC outputs a 2-cycle high pulse through the S_RST output signal. The 16-cycle sequence keeps repeating; after going from interrupt source cycle count of 0, 1, 2, 3, 4, ... 15, the count immediately returns to 0, 1, 2, etc., with no S_CLK delays between cycle count 15 and the next cycle count 0. Each time the sequence count is pointing to interrupt source 0, the $\overline{S_FRAME}$ signal is active. $\overline{S_FRAME}$ is provided to guarantee synchronization between the MPC8240 EPIC unit and the serial interrupt source device.

Note that initially, interrupt source 0 is sampled at the fifth S_CLK rising edge after S_RST negates. Also, once S_RST is asserted, it is not asserted again until after an EPIC reset, and the EPIC unit is subsequently programmed to serial mode again.

12.6.2 Edge/Level Sensitivity of Serial Interrupts

The interrupt detection is individually programmable for each source to be edge- or level-sensitive by writing the sense and polarity bits of the vector/priority register of the particular interrupt source. Refer to Section 12.3.6.1, “Interrupt Pending Register (IPR)—Non-programmable,” and the serial vector/priority register description in Section 12.9.8.1, “Direct & Serial Interrupt Vector/Priority Regs (IVPRs, SVPRs),” for more edge/level sensitivity information.

Note that for level-sensitive interrupts there is a potential race condition between an EOI (end of interrupt) command for a specific interrupt source; and the sampling of the same specific interrupt source as inactive. Level-sensitive interrupts are cleared from an IPR only when sampled as inactive; therefore, a second interrupt for the same source may occur, although the specific interrupt has already been serviced.

Software can avoid this second interrupt by delaying the EOI command to the EPIC unit. Depending on the interrupt source device being serviced, one possible software method is to first clear the interrupt from the source before executing any other necessary read or write transactions to service the interrupt device. In any case, the delay should be no less than 16 serial clocks after clearing the interrupt at the source device.

12.6.3 Serial Interrupt Timing Protocol

Figure 12-3 shows the relative timing for the serial interrupt interface signals.

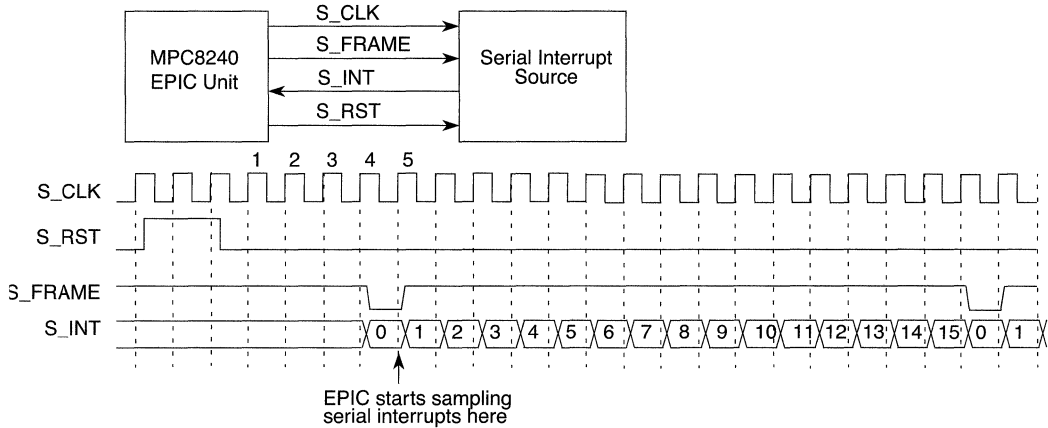


Figure 12-3. Serial Interrupt Interface Protocol

12.7 EPIC Timers

The MPC8240 has appropriate clock prescalers and synchronizers to provide a time base for the four global timers (0–3) of the EPIC unit. The global timers can be individually programmed to generate interrupts to the processor when they count down to zero and can be used for system timing or to generate regular periodic interrupts. Each timer has four registers for configuration and control:

- Global timer current count register (GTCCR)
- Global timer base count register (GTBCR)
- Global timer vector/priority register (GTVPR)
- Global timer destination register (GTDR)

The timers count at 1/8 the frequency of the SDRAM_CLK signals. The EPIC unit has a timer frequency reporting register (TFRR) that can be written by software to store the value of the timer frequency (as described in Table 12-11). Although this frequency is affected by the setting of the PLL_CFG[0–4] signals at reset, the system software must know the SDRAM_CLK frequency in order to accurately set this value. (There is no way to determine this frequency by simply reading an MPC8240 register.) The value written to TFRR does not affect the frequency of the timers.

Two of the timers, timer2 and timer3, can be set up to start automatically periodic DMA operations for DMA channels 0 and 1, respectively, without using the processor interrupt mechanism. In this case, the timer interrupt should be masked (GTVPR[M] = 1), and GTBCR[CI] should be cleared to start the counting. It is important to choose a rate for the timer longer than the time required to complete the DMA chain; otherwise, unpredictable

operation occurs. To complete the initialization of the periodic DMA feature, the DMA channel must be configured for chaining mode and the DMR[PDE] for the appropriate channel must be set. See Section 9.3.2.2, “Periodic DMA Feature.”

12.8 Programming Guidelines

Accesses to the EPIC unit include interrupt and timer initialization and reading the interrupt acknowledge register (IACK), which results in the EPIC unit returning the vector associated with the interrupt to be serviced. External interrupt sources IRQ0–4 can be programmed for either level- or edge-sensitive activation and either polarity. Similarly, all 16 serial interrupt sources can be programmed for either level- or edge-sensitive activation and for either polarity.

Most EPIC control and status registers are readable and return the last value written. The exceptions to this rule are as follows:

- EOI register, which returns zeros on reads
- Activity bit (A) of the vector/priority registers, which returns the value according to the status of the current interrupt source
- IACK register, which returns the vector of highest priority that is currently pending, or the spurious vector.
- Reserved bits, which always return 0

Even though reserved fields return 0, this should not be assumed by the programmer. Reserved bits should always be written with the value they returned when read. Thus the registers with reserved fields should be programmed by reading the value, modifying the appropriate fields, and writing back the value.

The following guidelines are recommended when the EPIC unit is programmed in mixed mode (GCR[M] = 1):

- If the processor’s memory management unit (MMU) is enabled, all EPIC registers must be located in a cache-inhibited area.
- The EPIC portion of the embedded utilities memory block (EUMB) must be set up appropriately. (Registers within the EUMB are located from 0x8000_0000 to 0xFDFD_FFFF.)
- The EPIC registers are described in this chapter in little-endian format. If the system is in big-endian mode, the bytes must be appropriately swapped by software.

In addition, the following sequence is recommended:

1. The vector, priority, and polarity values in each vector/priority register must have the mask (M) bit set initially.
2. The processor current task priority register (PCTPR) value must be set.
3. If serial mode is to be used, the EICR[SIE] must be set.
4. The M bit in the vector/priority registers to be used should then be cleared.
5. Depending on the interrupt system configuration, the EPIC unit may generate spurious interrupts to clear out interrupts latched during power-up. A spurious or non-spurious vector will be returned for an interrupt acknowledge cycle. See programming note below for the non-spurious vector case.

Programming Note:

Because edge-sensitive interrupts are not cleared until they are acknowledged and the default polarity/sense bits for all interrupts are set to edge-sensitive, it is possible for EPIC to store detections of edges at power-up as pending interrupts. If software permanently sets the polarity/sense of an interrupt source to edge-sensitive, it may receive the vector for the interrupt source and not a spurious vector once software clears the mask bit. This can occur once for any edge-sensitive interrupt source when its mask is first disabled and EPIC is in mixed mode.

To prevent having to handle a false interrupt for this case, software can clear the EPIC interrupt pending register of edges detected during power-up by first setting the polarity/sense bits of the interrupt source to level-sensitive; high level if the line is a positive-edge source, low level if the line is a negative-edge source (and the mask bit should remain set). Software can then set the interrupt source's polarity/sense bits to the appropriate values.

12.9 Register Definitions

The following sections describe the registers of the EPIC unit.

12.9.1 Feature Reporting Register (FRR)

The feature reporting register (FRR) provides information about the interrupt and processor configurations. It also contains controller version information. Note that this register is read-only. Figure 12-4 shows the bits in the FRR.

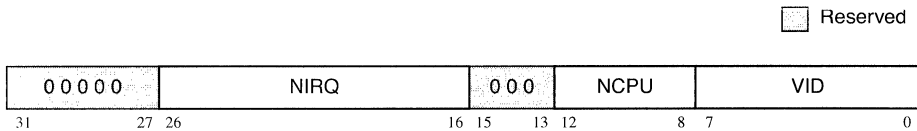


Figure 12-4. Feature Reporting Register (FRR)

Table 12-5 describes the bit settings for the FRR.

Table 12-5. FRR Field Descriptions— Offset 0x4_1000

Bits	Name	Reset Value	Description
31–27	—	All 0s	Reserved
26–16	NIRQ	0x017	Number of interrupts—This field contains the maximum number of interrupt sources supported. In the MPC8240, there are a maximum of 24 interrupts in use at one time: the 4 internal sources (I ² C, DMA (2), and I ₂ O), 4 timer sources and 16 external sources. A zero in this field corresponds to one interrupt, and so on. Thus the value of 0x017 corresponds to 24 interrupts.
15–13	—	All 0s	Reserved
12–8	NCPU	0x00	Number of CPUs—There is one CPU supported by the MPC8240: CPU 0.
7–0	VID	0x02	Version ID for this interrupt controller—This value reports the level of OpenPIC specification supported by this implementation. VID =2, representing version level 1.2 of OpenPIC, for the initial release of the MPC8240.

12.9.2 Global Configuration Register (GCR)

The GCR provides programming control resetting the EPIC unit and the external interrupts cascade mode. Note that this register is read/write. Figure 12-5 shows the bits in the GCR.

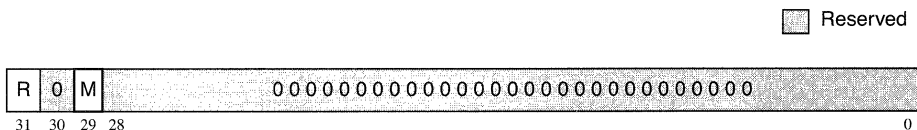


Figure 12-5. Global Configuration Register (GCR)

Table 12-6 describes the bit settings for the GCR.

Table 12-6. GCR Field Descriptions— Offset 0x4_1020

Bits	Name	Reset Value	Description
31	R	0	Reset EPIC unit. Writing a one to this bit resets the EPIC controller logic. This bit is cleared automatically when this reset sequence is complete. Setting this bit causes the following: <ul style="list-style-type: none"> • All pending and in-service interrupts are cleared • All interrupt mask bits are set • All timer current count values are reset to the base count • The processor current task priority is reset to 0xF thus disabling interrupt delivery to the processor. • Spurious vector resets to 0xFF • EPIC defaults to pass-through mode • The serial clock ratio resets to 0x4. All other registers remain at their pre-reset programmed values.
30	—	0	Reserved
29	M	0	Mode 0 Pass-through mode—EPIC is disabled and interrupts detected on IRQ0 (active-high) are passed directly to the processor core. 1 Mixed-mode—direct or serial. When this bit is set, EICR[SIE] determines whether the EPIC unit is operating in direct or serial interrupts mode.
28–0	—	All 0s	Reserved

12.9.3 EPIC Interrupt Configuration Register (EICR)

The EICR provides programming control for the serial interrupt mode and the serial clock frequency. Note that this register is read/write. Figure 12-6 shows the bits in the EICR.

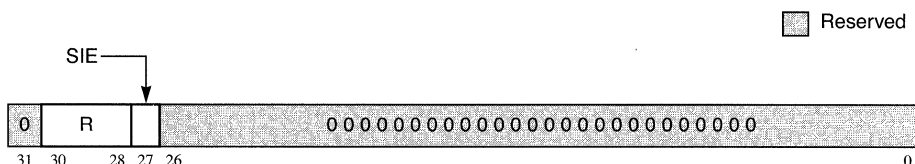


Figure 12-6. EPIC Interrupt Configuration Register (EICR)

Table 12-7 describes the bit settings for the EICR.

Table 12-7. EICR Field Descriptions—Offset 0x4_1030

Bits	Name	Reset Value	Description
31	—	0	Reserved
30–28	R	0x4	<p>Clock ratio—The S_CLK signal is driven by EPIC at a frequency of the SDRAM_CLK frequency divided by twice the value of this 3-bit field. The reset value of this field is 0x4. At this value, the S_CLK signal operates at 1/8th the frequency of the SDRAM_CLK signal. The allowable range of values for this field is between 1 and 7 resulting in a clock division ratio between 2 and 14 respectively.</p> <p>Warning: an illegal value could result in spurious vectors returned when in either direct or serial mode.</p>
27	SIE	0	<p>Serial interrupt enable—This bit selects whether the MPC8240 IRQ signals are configured for direct interrupts or serial interrupts. The GCR[M] must be set to 1 (mixed-mode) in order for this bit value to have meaning.</p> <p>0 direct interrupts mode 1 Serial interrupts mode</p>
26–0	—	All 0s	Reserved

12.9.4 EPIC Vendor Identification Register (EVI)

The EVI has specific read-only information about the vendor and the device revision. Figure 12-7 shows the bits in the EVI.

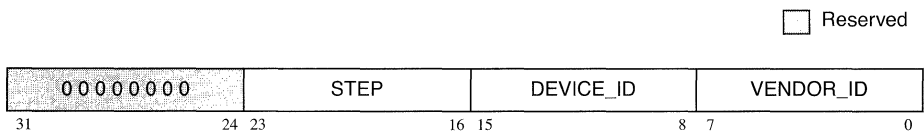


Figure 12-7. EPIC Vendor Identification Register (EVI)

Table 12-8 describes the bit settings for the EVI.

Table 12-8. EVI Register Field Descriptions— Offset 0x4_1080

Bits	Name	Reset Value	Description
31–24	—	All 0s	Reserved
23–16	STEP	0x01	Stepping—This indicates the stepping (silicon revision) for this device.
15–8	DEVICE_ID	All 0s	Device identification
7–0	VENDOR_ID	All 0s	Vendor identification—Because this value is zeros, the MPC8240 is considered to be a generic PIC-compliant device

12.9.5 Processor Initialization Register (PI)

The processor initialization register (PI) provides a mechanism for the software, through the EPIC unit, to cause a soft reset of the processor. Note that this register is read/write. Figure 12-8 shows the bits in the PI.

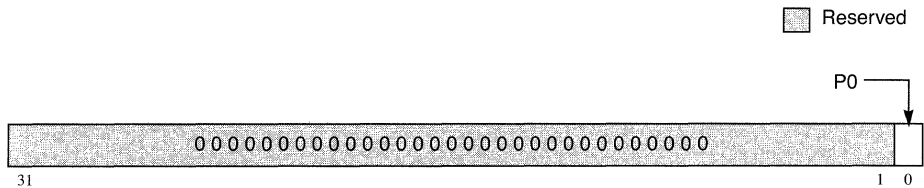


Figure 12-8. Processor Initialization Register (PI)

Table 12-9 describes the bit settings for the PI.

Table 12-9. PI Register Field Descriptions— Offset 0x4_1090

Bits	Name	Reset Value	Description
31-1	—	All 0s	Reserved
0	P0	0	Processor 0 soft reset. 0 default value 1 Setting this bit causes the EPIC unit to assert the internal \overline{sreset} signal to the processor core, causing a soft reset exception. The \overline{sreset} signal is edge-sensitive to the processor, but it is held active until a zero is written to P0. Thus, it should be cleared by software as soon as possible in the soft reset exception handler.

12.9.6 Spurious Vector Register (SVR)

The spurious vector register contains the 8-bit vector returned to the processor during an interrupt acknowledge cycle for the cases described in Section 12.3.5, “Spurious Vector Generation.” Note that this register is read/write. Figure 12-9 shows the bits in the SVR.

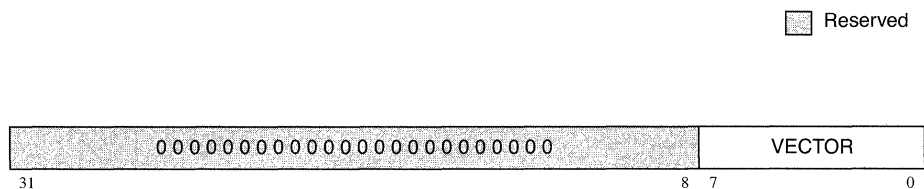


Figure 12-9. Spurious Vector Register (SVR)

Table 12-10 describes the bit settings for the SVR.

Table 12-10. SVR Field Descriptions— Offset 0x4_10E0

Bits	Name	Reset Value	Description
31–8	–	All 0s	Reserved
7–0	VECTOR	0xFF	Spurious interrupt vector—The vector value in this field is returned when the interrupt acknowledge register (IACK) is read during a spurious vector fetch.

12.9.7 Global Timer Registers

This section describes the global timer registers. Note that each of the four timers (timer0–timer3) has four individual configuration registers ($GTCCR_n$, $GTBCR_n$, $GTVPR_n$, $GTDR_n$), but they are shown only once in this section.

12.9.7.1 Timer Frequency Reporting Register (TFRR)

The TFRR is written by software to report the clocking frequency of the EPIC timers. Note that although this register is read/write, the value in this register is ignored by the EPIC unit. Figure 12-10 shows the bits in the TFRR.

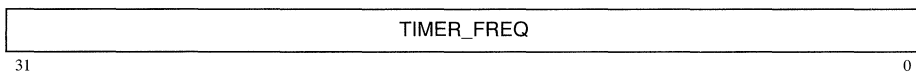


Figure 12-10. Timer Frequency Reporting Register (TFRR)

Table 12-11 describes the bit settings for the TFRR.

Table 12-11. TFRR Field Descriptions— Offset 0x4_10F0

Bits	Name	Reset Value	Description
31–0	TIMER_FREQ	All 0s	<p>Timer frequency—This register is used to report the frequency of the clock source for the global timers (in ticks/seconds (Hz)) which is always the SDRAM_CLK signal. The timers operate at 1/8th the speed of the SDRAM_CLK signal.</p> <p>The register is only set by software. The value in this register does not affect the speed of the timers. The timers' speed is determined by the PLL_CFG[0–4] signals and the frequency of the PCI_SYNC_IN signal. The value may be written by the system initialization code after the SDRAM_CLK frequency has been determined by the firmware. The firmware can use information stored in the HID1 register and information about the actual processor frequency to determine the SDRAM_CLK frequency. However, in some cases, more system frequency information may be required.</p>

12.9.7.2 Global Timer Current Count Registers (GTCCRs)

The GTCCRs contain the current count for each of the four EPIC timers. Note that these registers are read-only. Figure 12-11 shows the bits of the GTCCRs.

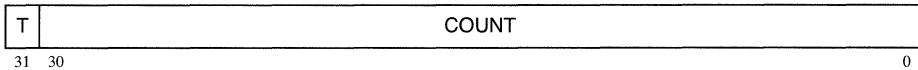


Figure 12-11. Global Timer Current Count Register (GTCCR)

Table 12-12 describes the bit settings for the GTCCRs. The address offset from EUMBBAR for GTCCRs is—0x4_1100, 0x4_1140, 0x4_1180, 0x4_11C0.

Table 12-12. GTCCR Field Descriptions

Bits	Name	Reset Value	Description
31	T	0	Toggle. This bit toggles whenever the current count decrements to zero.
30–0	COUNT	All 0s	Current timer count. This 31-bit field is decremented while the GTBCR[CI] bit is zero. When the timer counts down to zero, this field is reloaded from the base count register, the toggle bit is inverted, and an interrupt is generated (provided it is not masked).

12.9.7.3 Global Timer Base Count Registers (GTBCRs)

The GTBCRs contain the base count for each of the four EPIC timers. This is the value reloaded into the GTCCRs when they count down to zero. Note that these registers are read/write. Figure 12-12 shows the bits of the GTBCRs.

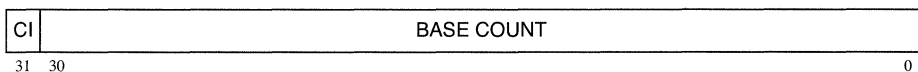


Figure 12-12. Global Timer Base Count Register (GTBCR)

Table 12-13 describes the bit settings for the GTBCRs. The Address offset from EUMBBAR for the GTBCRs is —0x4_1100, 0x4_1150, 0x4_1190, 0x4_11D0.

Table 12-13. GTBCR Field Descriptions

Bits	Name	Reset Value	Description
31	CI	1	Count inhibit. 0 Enables counting for this timer to proceed. 1 Inhibits counting for this timer.
30–0	BASE_COUNT	All 0s	Base count. This 31-bit field contains the base count used for this timer. When a value is written into this register and the CI bit transitions from a 1 to a 0, the base count value is copied into the corresponding current count register and the toggle bit is cleared.

12.9.7.4 Global Timer Vector/Priority Registers (GTVPRs)

The GTVPRs contain the interrupt vector and the interrupt priority for each of the four timers. In addition, they contain the mask and activity bits for each of the four timers. Note that these registers are read/write. Figure 12-13 shows the bits of the GTVPRs.

 Reserved



Figure 12-13. Global Timer Vector/Priority Register (GTVPR)

Table 12-14 describes the bit settings for the GTVPRs. The Address Offset from EUMBBAR for the GTVPR is —0x4_1120, 0x4_1160, 0x4_11A0, 0x4_11F0.

Table 12-14. GTVPR Field Descriptions

Bits	Name	Reset Value	Description
31	M	1	Mask—Mask interrupts from this timer 0 If the mask bit is cleared while the corresponding IPR bit is set, $\overline{\text{int}}$ is asserted to the processor. 1 Further interrupts from this timer are disabled
30	A	0	Activity—Indicates that an interrupt has been requested or that it is in-service. Note that this bit is read-only. 0 No current interrupt activity associated with this timer. 1 The interrupt bit for this timer in the IPR or ISR is set. The VECTOR and PRIORITY values should not be changed while the A bit is set.
29–20	—	All 0s	Reserved.

Table 12-14. GTVPR Field Descriptions (Continued)

Bits	Name	Reset Value	Description
19–16	PRIORITY	0x0	Priority—This field contains the four-bit interrupt priority. The lowest priority is 0 and the highest is 15. A priority level of 0 disables interrupts from this timer.
15–8	—	0x00	Reserved
7–0	VECTOR	0x00	Vector—The vector value in this field is returned when the interrupt acknowledge register (IACK) is read, and the interrupt associated with this vector has been requested.

12.9.7.5 Global Timer Destination Registers (GTDRs)

Each GTDR indicates the destination for the timer’s interrupt. Because the MPC8240 is a single-processor device, the destination is always P0. Note that this register is read-only. Figure 12-14 shows the bits of the GTDRs.

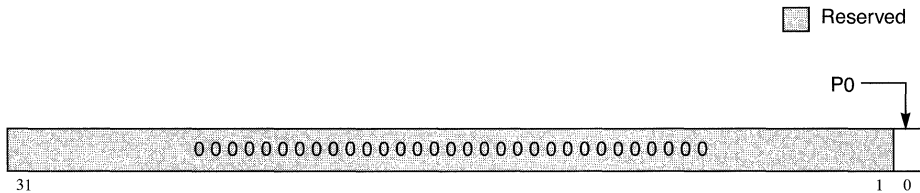


Figure 12-14. Global Timer Destination Register (GTDR)

Table 12-15 describes the bit settings for the GTDRs. The Address Offset from EUMBBAR for GTDRs is—0x4_1130, 0x4_1170, 0x4_11B0, 0x4_11F0.

Table 12-15. GTDR Field Descriptions

Bits	Name	Reset Value	Description
31–1	—	All 0s	Reserved
0	P0	1	Processor 0—Timer interrupt always directed to the processor.

12.9.8 Direct, Serial, and Internal Interrupt Registers

This section describes the vector/priority and destination registers for the direct, serial, and internal (I²C, DMA, and I₂O) interrupt sources.

12.9.8.1 Direct & Serial Interrupt Vector/Priority Regs (IVPRs, SVPRs)

The format for the IRQ0–4 (direct) vector/priority registers (IVPRs) is identical to that of the vector/priority registers for the 16 serial interrupts (SVPRs). Note that these registers are read/write.

Figure 12-15 shows the bits of the IVPRs and SVPRs.

 Reserved



Figure 12-15. Direct and Serial Interrupt Vector/Priority Registers (IVPR and SVPR)

Table 12-16 shows the bit settings for the IVPRs and SVPRs. The Address Offset from EUMBBAR for IVPRs is—0x5_0200, 0x5_0220, 0x5_0240, 0x5_0260, 0x5_0280. The Address Offset from EUMBBAR for SVPRs —0x5_0200, 0x5_0220, 0x5_0240, 0x5_0260, 0x5_0280, 0x5_02A0, 0x5_02C0, 0x5_02E0, 0x5_0300, 0x5_0320, 0x5_0340.

Table 12-16. IVPR and SVPR Field Descriptions

Bits	Name	Reset Value	Description
31	M	1	Mask. Masks interrupts from this source. 0 If the mask bit is cleared while the corresponding IPR bit is set, $\overline{\text{int}}$ is asserted to the processor. 1 Further interrupts from this source are disabled
30	A	0	Activity. Indicates that an interrupt has been requested or that it is in-service. Note that this bit is read-only. 0 No current interrupt activity associated with this source. 1 The interrupt bit for this source in the IPR or ISR is set. The VECTOR, PRIORITY, P (polarity), or S (sense) values should not be changed while the A bit is set, except to clear an old interrupt.
29–24	—	All 0s	Reserved.
23	P	0	Polarity. This bit sets the polarity for the external interrupt. 0 Polarity is active-low or negative-edge triggered 1 Polarity is active-high or positive-edge triggered
22	S	0	Sense. This bit sets the sense for external interrupts. 0 The external interrupt is edge-sensitive. 1 The external interrupt is level-sensitive.
21–20	—	All 0s	Reserved.
19–16	PRIORITY	0x0	Priority. This field contains the four-bit interrupt priority. The lowest priority is 0 and the highest is 15. A priority level of 0 disables interrupts from this source.
15–8	—	0x00	Reserved
7–0	VECTOR	0x00	Vector. The vector value in this field is returned when the interrupt acknowledge register (IACK) is read and the interrupt associated with this vector has been requested.

12.9.8.2 Direct & Serial Interrupt Destination Registers (IDRs, SDRs)

The IDR and SDR registers indicate the destination for the each external interrupt source. Because the MPC8240 is a single-processor device, the destination is always P0. Note that this register is read-only. Figure 12-16 shows the bits of the IDRs and SDRs.

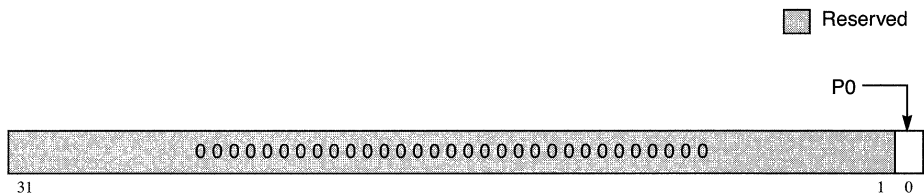


Figure 12-16. Direct and Serial Destination Registers (IDR and SDR)

Table 12-17 shows the bit settings for the IDRs and SDRs. The Address offset from EUMBBAR for IDRs is—0x5_0210, 0x5_0230, 0x5_0250, 0x5_0270, 0x5_0290. The Address offset from EUMBBAR for SDRs is—0x5_0210, 0x5_0230, 0x5_0250, 0x5_0270, 0x5_0290, 0x5_02B0, 0x5_02D0, 0x5_02F0, 0x5_0310, 0x5_0330, 0x5_0350, 0x5_0370, 0x5_0390, 0x5_03B0, 0x5_03D0, 0x5_03F0.

Table 12-17. IDR and SDR Field Descriptions

Bits	Name	Reset Value	Description
31-1	—	All 0s	Reserved
0	P0	1	Processor 0—direct and serial interrupts always directed to the processor.

12.9.8.3 Internal (I²C, DMA, I₂O) Interrupt Vector/Priority Regs (IIVPRs)

The IIVPRs have the same format and field descriptions as the GTVPRs, except that they apply to the internal MPC8240 interrupt sources (the I²C, DMA (2 channels) and I₂O units). See Section 12.9.7.4, “Global Timer Vector/Priority Registers (GTVPRs),” for a complete description of the GTVPRs.

12.9.8.4 Internal (I²C, DMA or I₂O) Interrupt Destination Regs (IIDRs)

The IIDRs have the same format and field descriptions as the IDRs (and SDRs), except that they apply to the internal MPC8240 interrupt sources (the I²C, DMA (2 channels) and I₂O units). See Section 12.9.8.2, “Direct & Serial Interrupt Destination Registers (IDRs, SDRs),” for a complete description of the IDRs.

12.9.9 Processor-Related Registers

This section describes the processor-related EPIC registers.

12.9.9.1 Processor Current Task Priority Register (PCTPR)

Software writes the priority of the current processor task in the PCTPR. The EPIC unit uses this value to compare with the priority of incoming interrupts. If an incoming interrupt has a greater priority than that assigned in the PCTPR, is greater than other incoming interrupts, and is not masked, the \overline{int} signal is asserted to the processor. Priority levels from 0 (lowest) to 15 (highest) are supported. Setting the task priority to 15 masks all interrupts to the processor. The PCTPR is initialized to 0x0000_000F when the MPC8240 is reset, or when the P0 bit of the processor initialization register is set to one. Note that this register is read/write. Figure 12-17 shows the bits of the PCTPR.

Reserved

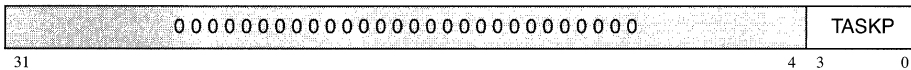


Figure 12-17. Processor Current Task Priority Register (PCTPR)

Table 12-18 shows the bit settings for the PCTPR.

Table 12-18. PCTPR Field Descriptions— Offset 0x6_0080

Bits	Name	Reset Value	Description
31–4	—	All 0s	Reserved
3–0	TASKP	0xF	Task priority—Set 0 to 15, where 15 corresponds to the highest priority for processor tasks. When PCTPR[TASKP] = 0xF, no interrupts will be signalled to the processor.

12.9.9.2 Processor Interrupt Acknowledge Register (IACK)

The interrupt acknowledge mechanism on the MPC8240 consists of a read from the memory-mapped interrupt acknowledge register (IACK) in the EPIC unit. Reading the IACK returns the interrupt vector corresponding to the highest priority pending interrupt. Reading IACK also has the following side effects:

- The associated bit in the IPR is cleared (if it is configured as edge-sensitive).
- The ISR is updated.
- The internal \overline{int} signal is negated.

Reading IACK when there is no interrupt pending returns the spurious vector value. Note that this register is read-only.

Register Definitions

Figure 12-18 shows the bits of the IACK.

 Reserved

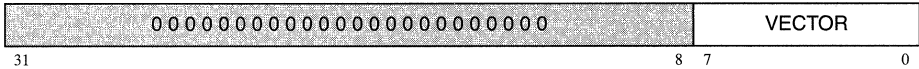


Figure 12-18. Processor Interrupt Acknowledge Register (IACK)

Table 12-19 shows the bit settings of the IACK.

Table 12-19. IACK Field Descriptions— Offset 0x6_00A0

Bits	Name	Reset Value	Description
31–8	–	All 0s	Reserved
7–0	VECTOR	0x0	Interrupt vector. When this register is read, this field returns the vector of the highest pending interrupt in the EPIC unit.

12.9.10 Processor End-of-Interrupt Register (EOI)

The write to the EOI signals the end of processing for the highest priority interrupt currently in service by the processor. The write to EOI updates the ISR by retiring the highest priority interrupt. EOI values other than 0 are undefined. Data values written to this register are ignored and zero is assumed. Reading this register returns zeros. Note that this register is write-only. Figure 12-19 shows the bits of the EOI.

 Reserved

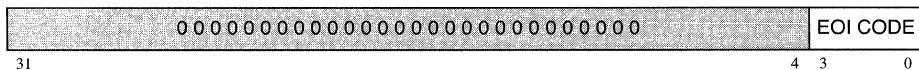


Figure 12-19. Processor End of Interrupt Register (EOI)

Table 12-20 shows the bit settings for the EOI.

Table 12-20. EOI Field Descriptions— Offset 0x6_00B0

Bits	Name	Reset Value	Description
31–4	–	–	Reserved
3–0	EOI_CODE	–	0000

Chapter 13

Error Handling and Exceptions

The MPC8240 provides error detection and reporting on the three primary interfaces (processor core interface, memory interface, and PCI interface). This chapter describes how the MPC8240 handles different error conditions.

13.1 Overview

Errors detected by the MPC8240 are reported to the processor core by asserting an internal machine check signal (\overline{mcp}). The state of the internal machine check signal is externally driven on the \overline{MCP} output signal. The system error (\overline{SERR}) and parity error (\overline{PERR}) signals are used to report errors on and to the PCI bus. The MPC8240 provides the NMI signal for ISA bridges to report errors on the ISA bus. The MPC8240 internally synchronizes any asynchronous error signals.

The PCI command, status, and error handling registers enable or disable the reporting and detection of specific errors. These registers are described in Chapter 5, “Configuration Registers.”

The MPC8240 detects illegal transfer types from the processor, illegal Flash write transactions, PCI address and data parity errors, accesses to memory addresses out of the range of physical memory, memory parity errors, memory refresh overflow errors, ECC errors, PCI master-abort cycles, and PCI received target-abort errors.

The MPC8240 latches the address and type of transaction that caused the error in the error status registers to assist diagnostic and error handling software. See Section 5.8, “Address Map B Options Register,” for more information. Section 3.2.5, “System Control and Power Management Signals,” contains the signal definitions for the interrupt signals.

13.1.1 Error Handling Block Diagram

Figure 13-1 provides the internal error management block diagram.

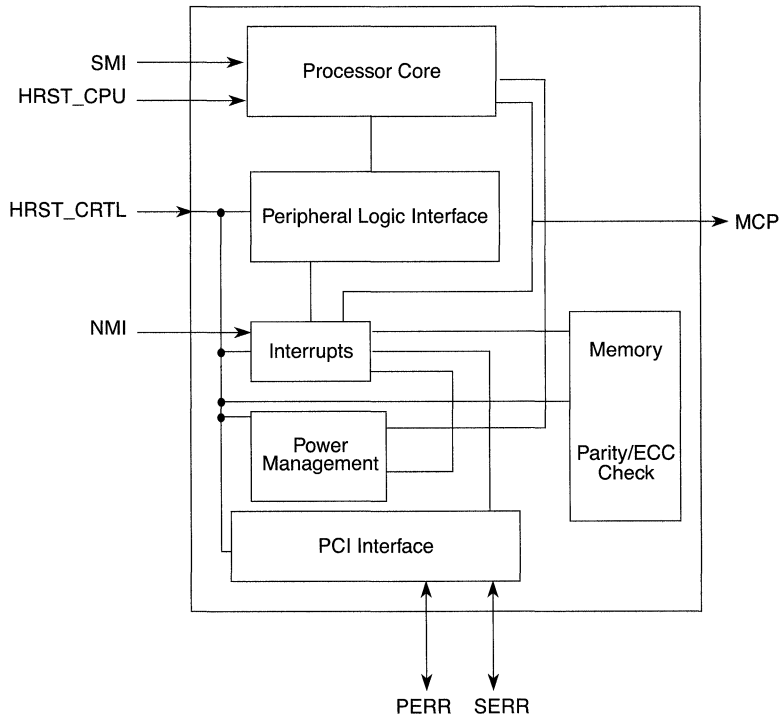


Figure 13-1. Internal Error Management Block Diagram

13.1.2 Priority of Externally-Generated Errors and Exceptions

Many of the errors detected in the MPC8240 cause exceptions to be taken by the processor core. Table 13-1 describes the relative priorities and recoverability of externally-generated errors and exceptions. The processor exception generated by each of these conditions is described in Section 2.5, “Exception Model.”

Table 13-1. MPC8240 Error Priorities

Priority	Exception	Cause
0	Hard reset	Hard reset (required on power-on); HRST_CTRL and HRST_CPU asserted (must always be asserted together)
1	Machine check	Processor transaction error, or Flash write error
2	Machine check	PCI address parity error (\overline{SERR}) or PCI data parity error (\overline{PERR}) when the MPC8240 is acting as the PCI target
3	Machine check	Memory select error, memory data read parity error, memory refresh overflow, or ECC error

Table 13-1. MPC8240 Error Priorities (Continued)

Priority	Exception	Cause
4	Machine check	PCI address parity error ($\overline{\text{SERR}}$) or PCI data parity error ($\overline{\text{PERR}}$) when the MPC8240 is acting as the PCI master, PCI master-abort, or received PCI target-abort
5	Machine check	NMI (nonmaskable interrupt)

Note that for priority 1 through 5, the exception is the same. The machine check exception and the priority are related to additional error information provided by the MPC8240 (for example, the address provided in the Processor/PCI error address register).

13.2 Exceptions and Error Signals

Although Section 3.2.5, “System Control and Power Management Signals,” contains the signal definitions for the exception and error signals, this section describes the interactions between system components when an exception or error signal is asserted.

13.2.1 System Reset

The system reset exception is an asynchronous, nonmaskable interrupt that occurs when the hard reset input signals are asserted (required at power-on). The MPC8240 has two hard reset input signals, $\overline{\text{HRST_CPU}}$ and $\overline{\text{HRST_CTRL}}$, and they must be asserted and negated simultaneously.

When a system reset is recognized ($\overline{\text{HRST_CPU}}$ and $\overline{\text{HRST_CTRL}}$ are both asserted), the MPC8240 aborts all current internal and external transactions, releases all bidirectional I/O signals to a high-impedance state, ignores the input signals (except for PCI_SYNC_IN and the configuration signals described in Section 3.4, “Configuration Pins Sampled at Reset”), and drives most of the output signals to an inactive state. Table 3-2 on page -7 shows the states of the output-only signals during system reset. The MPC8240 then initializes its internal logic.

For proper initialization, the assertion of $\overline{\text{HRST_CPU}}$ and $\overline{\text{HRST_CTRL}}$ must satisfy the minimum active pulse width requirements given in the MPC8240 Hardware Specification.

During system reset, the latches dedicated to JTAG functions are not initialized. The IEEE 1149.1 standard prohibits the device reset from resetting the JTAG logic. The JTAG reset ($\overline{\text{TRST}}$) signal is required to reset the dedicated JTAG logic during power-on.

13.2.2 Processor Core Error Signal ($\overline{\text{mcp}}$)

The MPC8240 provides an internal machine check signal ($\overline{\text{mcp}}$) to the processor core for error reporting. The internal machine check signal indicates to the processor that a nonrecoverable error has occurred during system operation. The state of $\overline{\text{mcp}}$ is provided externally on the $\overline{\text{MCP}}$ output signal. The assertion of $\overline{\text{mcp}}$ depends upon whether the error handling registers of the MPC8240 are set to report the specific error. The programmable

Exceptions and Error Signals

parameter PICR1[MCP_EN] is used to enable or disable the assertion of \overline{mcp} by the MPC8240 for all error conditions. Assertion of \overline{mcp} causes the processor core to take a machine check exception conditionally or enter the checkstop state based on the value of MSR[ME].

The machine check signal may be asserted to the processor core on any cycle. Whether the current transaction is aborted depends upon the software configuration.

The MPC8240 holds \overline{mcp} asserted until the processor core has taken the exception. The MPC8240 decodes a machine check acknowledge cycle by detecting processor reads from the two possible machine check exception addresses at 0x0000_0200–0x0000_0207 and 0xFFFF0_0200–0xFFFF0_0207; and then negates \overline{mcp} .

13.2.3 PCI Bus Error Signals

The MPC8240 uses three error signals to interact with the PCI bus— \overline{SERR} , \overline{PERR} , and NMI.

13.2.3.1 System Error (\overline{SERR})

The \overline{SERR} signal is used to report PCI address parity errors, PCI data parity errors on a special-cycle command, target-abort, or any other errors where the result is potentially catastrophic. The \overline{SERR} signal is also asserted for master-abort, except for PCI configuration accesses or special-cycle transactions.

The agent responsible for driving AD[31–0] on a given PCI bus phase is responsible for driving even parity one PCI clock cycle later on the PAR signal. That is, the number of 1s on AD[31–0], $\overline{C/BE}$ [3–0], and PAR equals an even number.

The \overline{SERR} signal is driven for a single PCI clock cycle by the agent that is reporting the error. The target agent is not allowed to terminate with retry or disconnect if \overline{SERR} is activated due to an address parity error.

Bits 8 and 6 of the PCI command register control whether the MPC8240 asserts \overline{SERR} upon detecting one of the error conditions. Bit 14 of the PCI status register reports when the MPC8240 has asserted the \overline{SERR} signal.

13.2.3.2 Parity Error (\overline{PERR})

The \overline{PERR} signal is used to report PCI data parity errors during all PCI transactions, except for PCI special-cycle command transactions. The agent responsible for driving AD[31–0] on a given PCI bus phase is responsible for driving even parity one PCI clock cycle later on the PAR signal. That is, the number of 1s on AD[31–0], $\overline{C/BE}$ [3–0], and PAR equals an even number.

Two PCI clock cycles after the data phase for which a data parity error is detected, the \overline{PERR} signal must be asserted by the agent receiving the data. Only the master may report a read data parity error; and only the selected target may report a write data parity error.

Bit 6 of the PCI command register decides whether the MPC8240 ignores $\overline{\text{PERR}}$. Bit 15 and bit 8 of the PCI status register are used to report when the MPC8240 has detected or reported a data parity error.

13.2.3.3 Nonmaskable Interrupt (NMI)

The NMI signal is, effectively, a PCI sideband signal between the PCI-to-ISA bridge and the MPC8240. The NMI signal is usually driven by the PCI-to-ISA bridge to report any nonrecoverable error detected on the ISA bus (normally, through the $\overline{\text{IOCHCK}}$ signal on the ISA bus). The name nonmaskable interrupt is misleading due to its history in ISA bus designs. The NMI signal should be connected to GND if it is not used. If $\text{PICR1}[\text{MCP_EN}]$ is set, the MPC8240 reports the NMI error to the processor core by asserting *mcp*.

13.3 Error Reporting

Error detection on the MPC8240 is designed to log the occurrence of an error and also log information related to the error condition. The individual error detection bits are contained in the PCI status register, error detection register 1 (ErrDR1), and error detection register 2 (ErrDR2). These bits indicate which error has been detected. (The error detection bits are specifically bits 15, 13, and 12 in the PCI status register, bits 7–4 and 2–0 in ErrDR1, and bits 5–3 and 0 in ErrDR2.)

The intent of error reporting is to log the information pertaining to the first error that occurs and prevent additional errors from being reported until the first error is acknowledged (and cleared). For additional errors to be reported, all error detection bits must be cleared. When an error detection bit is set, the MPC8240 does not report additional errors until all of the error detection bits are cleared. Note that more than one of the error detection bits can be set if simultaneous errors are detected. Therefore, software must check whether more than one bit is set before trying to determine information about the error.

The processor/PCI error address register, the processor bus error status register, and the PCI bus error status register together with ErrDR1[3] (processor/PCI cycle) and ErrDR2[7] (invalid error address) provide additional information about a detected error condition. When an error is detected, the associated information is latched inside these registers until all error detection bits are cleared. Subsequent errors set the appropriate error detection bits, but the bus error status and error address registers retain the information for the initial error until all error detection bits are cleared.

As described in Section 13.2.2, “Processor Core Error Signal (*mcp*),” the MPC8240 asserts *mcp* to the processor core when an enabled error condition has occurred during system operation. The assertion of *mcp* depends upon whether the error handling registers of the MPC8240 are set to report the specific error. Once asserted, the MPC8240 continues to assert *mcp* until the MPC8240 decodes a read from the processor to the machine check exception vector (0xnnn0_0200). When it decodes a processor read from the machine check exception vector, the MPC8240 negates *mcp*. However, until all the error detection bits are cleared, the MPC8240 does not report subsequent errors by reasserting *mcp*.

Error Reporting

In addition to the error detection bits, the MPC8240 reports the assertion of NMI to the processor core by asserting \overline{mcp} (if enabled). Note that NMI assertion is not recorded in the MPC8240's error detection bits. Reporting NMI assertion (by \overline{mcp}) can be masked by any error detection bits that are set.

13.3.1 Processor Interface

The processor interface of the MPC8240 detects unsupported processor bus transaction errors and Flash write errors. In these cases, both ErrDR1[3] and ErrDR2[7] are cleared, indicating that the error is due to a processor transaction and the address in the processor/PCI error address register is valid. Internally, the MPC8240 asserts transfer acknowledge (provided PICR1[10] = 0) to terminate the data tenure.

13.3.1.1 Processor Transaction Error

When a processor transaction error occurs, ErrDR1[1–0] is set to reflect the error type. Unsupported processor bus transactions include writes to the PCI interrupt-acknowledge space (0xBFFF_FFFn using address map A or 0xFEFn_nnnn using address map B), and attempts to execute the graphic read or graphic write instructions (**eciwx** or **ecowx**).

13.3.1.2 Flash Write Error

The MPC8240 allows writes to the system ROM space when PICR1[FLASH_WR_EN] is set and PICR2[FLASH_WR_LOCKOUT] is cleared. Otherwise, any processor write transaction to the system ROM space results in a Flash write error. When a Flash write error occurs, ErrDR2[0] is set.

The ROM/Flash interface on the MPC8240 accommodates only single-beat, data-path sized (8-, 32-, or 64-bit depending on the configuration) writes to Flash memory. Software must partition larger data into individual data path sized (8-, 32-, or 64-bit) write operations. However, attempts to write to Flash with a data size other than the full data path size will not cause a Flash write error.

13.3.2 Memory Interface

The memory interface of the MPC8240 detects read parity, ECC, memory select, and refresh overflow errors. The MPC8240 detects parity errors on the data bus during memory (DRAM/EDO/SDRAM) read cycles. When MCCR2[ECC_EN] is set, the memory controller can detect single-bit and multi-bit errors for system memory read transactions. Since the ECC logic corrects single-bit errors, they are reported only when the number of errors in the ECC single-bit error counter register equals the threshold value in the ECC single-bit error trigger register. A memory select error occurs when a system memory transaction address falls outside of the physical memory boundaries. A refresh overflow error occurs when no refresh transaction occurs within the equivalent of 16 refresh cycles.

In all cases, if the memory transaction is initiated by a PCI master, ErrDR1[3] is set; if the memory transaction is initiated by the processor core, ErrDR1[3] is cleared.

ErrDR2[7] is cleared to indicate that the error address in the processor/PCI error address register is valid. If the ECC single-bit error trigger threshold is reached, then the error address indicates the address of the most recent ECC single-bit error. Note that when a parity or ECC error occurs on the last beat of a transaction and another transaction to the same page has started, the MPC8240 cannot provide the error address and the corresponding bus status. In these cases, ErrDR2[7] is set to indicate that the error address in the processor/PCI error address register is not valid. The MPC8240 cannot provide the error address and the bus status for refresh overflow errors, so ErrDR2[7] is set for these errors as well.

If the transaction is initiated by the processor core or by a PCI master with bit 6 of the PCI command register cleared, the error status information is latched, but the transaction continues and terminates normally.

13.3.2.1 System Memory Read Data Parity Error

When MCCR1[PCKEN] is set, the MPC8240 checks memory parity on every memory read cycle and generates the parity on every memory write cycle that emanates from the MPC8240. When a read parity error occurs, ErrDR1[2] is set.

The MPC8240 does not check parity for transactions in the system ROM address space. Note that the processor should not check parity for system ROM space transactions as the parity data will be incorrect for these accesses.

13.3.2.2 System Memory ECC Error

When MCCR2[ECC_EN] is set, the MPC8240 performs an ECC check on every memory read cycle and generates the ECC check data on every memory write cycle. When a single-bit ECC error occurs, the ECC single-bit error counter register is incremented by 1 and its value is compared to the value in the ECC single-bit error trigger register. If the values are equal, ErrDR1[2] is set. In addition to single-bit errors, the MPC8240 detects all 2-bit errors, all errors within a nibble (one-half byte), and any other multibit error that does not alias to either a single-bit error or no error. When a multibit ECC error occurs, ErrDR2[3] is set.

When configured for in-line ECC with SDRAM, the MPC8240 cannot report ECC single-bit errors, see Section 6.4.11, “SDRAM and In-Line ECC or Parity,” for more information. Write parity errors are reported in ErrDR1[2] (memory read parity error/ECC single-bit error exceeded). Read parity and multiple-bit ECC errors are reported in ErrDR2[3] (ECC multi-bit error).

13.3.2.3 System Memory Select Error

A memory select error occurs when the address for a system memory transaction falls outside of the programmed boundaries of physical memory. When a memory select error occurs, ErrDR1[5] is set. If a write transaction causes a memory select error, the write data is simply ignored. If a read transaction causes the memory select error, the MPC8240 returns 0xFFFF_FFFF (all 1s). No $\overline{\text{RAS}}$ / $\overline{\text{CS}}$ signals are asserted in either case.

13.3.2.4 System Memory Refresh Overflow Error

When there are no refresh transactions for a period equal to 16 refresh cycles, the MPC8240 reports the error as a refresh overflow. When the MPC8240 detects a refresh overflow, ErrDR1[3] is set.

13.3.3 PCI Interface

The MPC8240 supports the error detection and reporting mechanism as specified in the PCI Local Bus Specification, Revision 2.1. The MPC8240 keeps error information and sets the appropriate error flags when a PCI error occurs (provided the corresponding enable bit is set), independent of whether the PCI command register is programmed to respond to or detect the specific error.

In cases of PCI errors, ErrDR1[3] is set to indicate that the error is due to a PCI transaction. In most cases, ErrDR2[7] is cleared to indicate that the error address in the processor/PCI error address register is valid. In these cases, the error address is the address as seen by the PCI bus, not the processor core physical address.

If NMI is asserted, the MPC8240 cannot provide the error address and the corresponding bus error status. In such cases, ErrDR2[7] is set to indicate that the error address in the processor/PCI error address register is not valid.

13.3.3.1 Address Parity Error

If the MPC8240 is acting as a PCI master, and the target detects and reports (by asserting $\overline{\text{SERR}}$) a PCI address parity error, then the MPC8240 sets ErrDR1[7] and sets the detected parity error bit (bit 15) in the PCI status register. This is independent of the settings in the PCI command register. Note that for the MPC8240 to recognize the assertion of SERR by another PCI agent, ErrEnR1[7] (RX_SERR_EN) must be set.

If the MPC8240 is acting as a PCI target and detects a PCI address parity error, the PCI interface of the MPC8240 sets the status bit in the PCI status register (bit 15). If bits 8 and 6 of the PCI command register are set, the MPC8240 reports the address parity error by asserting $\overline{\text{SERR}}$ to the master (two clocks after the address phase) and sets bit 14 of the PCI status register. Also, if PICR1[MCP_EN] is set, the MPC8240 reports the error to the processor core by asserting \overline{mcp} .

13.3.3.2 Data Parity Error

If the MPC8240 is acting as a PCI master and a data parity error occurs, the MPC8240 sets bit 15 of the PCI status register. This is independent of the settings in the PCI command register.

If the PCI command register of the MPC8240 is programmed to respond to parity errors (bit 6 of the PCI command register is set) and a data parity error is detected or signaled during a PCI bus transaction, the MPC8240 sets the appropriate bits in the PCI status register (bit 15 is set, and possibly bit 8 is set, as described in the following paragraphs).

If a data parity error is detected by the MPC8240 acting as the master (for example, during a processor-read-from-PCI transaction), and bit 6 of the PCI command register is set, the MPC8240 reports the error to the PCI target by asserting $\overline{\text{PERR}}$ and by setting bit 8 of the status register and tries to complete the transaction, if possible. Also, if $\text{PICR1}[\text{MCP_EN}]$ is set, the MPC8240 asserts $\overline{\text{mcp}}$ to report the error to the processor core. These actions also occur if the MPC8240 is the master and detects the assertion of $\overline{\text{PERR}}$ by the target (for a write).

If the MPC8240 is acting as a PCI target when the data parity error occurs (on a write), the MPC8240 asserts $\overline{\text{PERR}}$ and sets $\text{ErrDR1}[6]$ (PCI target $\overline{\text{PERR}}$). If the data has been transferred, the MPC8240 completes the operation but discards the data. Also, if $\text{PICR1}[\text{MCP_EN}]$ is set, the MPC8240 asserts $\overline{\text{mcp}}$ to report the error to the processor core. If the master asserts $\overline{\text{PERR}}$ during a memory read, the address of the transfer is logged in the error address register and $\overline{\text{mcp}}$ is asserted (if enabled).

13.3.3.3 Master-Abort Transaction Termination

If the MPC8240, acting as a master, initiates a PCI bus transaction (excluding special-cycle transactions), but there is no response from any PCI agent (DEVSEL has not been asserted within five PCI bus clocks from the start of the address phase), the MPC8240 terminates the transaction with a master-abort and sets the master-abort flag (bit 13) in the PCI status register. Special-cycle transactions are normally terminated with a master-abort, but these terminations do not set the master-abort flag in the PCI status register.

If $\text{ErrEnR1}[1]$ and $\text{PICR1}[\text{MCP_EN}]$ are both set and the MPC8240 terminates a transaction with a master-abort, the MPC8240 reports the error to the processor core by asserting $\overline{\text{mcp}}$.

13.3.3.4 Received Target-Abort Error

If a PCI transaction initiated by the MPC8240 is terminated by target-abort, the received target-abort flag (bit 12) of the PCI status register is set. If $\text{ErrEnR2}[1]$ and $\text{PICR1}[\text{MCP_EN}]$ are both set and the MPC8240 receives a target-abort, the MPC8240 reports the error to the processor core by asserting $\overline{\text{mcp}}$.

Note that any data transferred in a target-aborted transaction may be corrupt.

13.3.3.5 NMI (Nonmaskable Interrupt)

If $\text{PICR1}[\text{MCP_EN}]$ is set and a PCI agent asserts the NMI signal to the MPC8240, the MPC8240 reports the error to the processor core by asserting $\overline{\text{mcp}}$.

When the NMI signal is asserted, no error flags are set in the status registers of the MPC8240. The agent that drives NMI should provide the error flag for the system and the mechanism to reset that error flag. The NMI signal should then remain asserted until the error flag is cleared.

13.4 Exception Latencies

Latencies for taking various exceptions depend on the state of the MPC8240 when the conditions to produce an exception occur. The minimum latency is one cycle, in which case the exception is signaled in the cycle after the exception condition occur.

Chapter 14

Power Management

A key feature of the MPC8240 and its predecessor, the MPC603e microprocessor, is that they were designed for low-power operation. The MPC8240 provides both automatic and program-controllable power reduction modes for progressive reduction of power consumption. This chapter describes the hardware support features provided by the MPC8240 for power management of both the processor core and the peripheral logic.

14.1 Overview

The MPC8240 has explicit power management features for both the processor core and the peripheral logic both of which are described in this chapter. Note that the design of the MPC8240 is fully static, allowing the internal logic states to be preserved during power management. The MPC8240 implementation offers the following enhancements to the original MPC603e family:

- Lower-power design
- 2.5-volt core and 3.3-volt I/O

Because operating systems service I/O requests by system calls to the device drivers, the device drivers must be modified for power management. When a device driver is called to reduce the power of a device, it needs to be able to check the power state of the device, save the device configuration parameters, and put the device into a power-saving mode. Furthermore, every time the device driver is called it needs to check the power status of the device; and restore the device to the full-on state, if the device is in a power-saving mode.

14.2 Processor Core Power Management

The processor core has various types of power management features. Dynamic power management occurs automatically (if enabled), depending on the activity of the execution units. The programmable doze, nap, and sleep modes provide further power savings. The power management features in the processor core are very similar to that in the MPC603e device.

14.2.1 Dynamic Power Management

Dynamic power management (DPM) automatically powers up and down the individual execution units of the MPC8240 processor core, based upon the contents of the instruction stream. For example, if no floating-point instructions are being executed, the floating-point unit is automatically powered down. Power is not actually removed from the execution unit; instead, each execution unit has an independent clock input that is automatically controlled on a clock-by-clock basis. Because CMOS circuits consume negligible power when they are not switching, stopping the clock to an execution unit effectively eliminates its power consumption. The operation of DPM is transparent to software or any external hardware. Dynamic power management is enabled by setting bit 11 in `HID0` following a hard reset sequence.

14.2.2 Programmable Power Modes on Processor Core

The peripheral logic block can wake up the processor from a low power state through the internal asynchronous interrupt (\overline{int}) signal (generated by the EPIC unit), which transfers program flow to the interrupt handler code. The appropriate mode then is set by the software. The MPC8240 provides a second interrupt signal and interrupt vector for power management—the system management interrupt (\overline{SMI}). The MPC8240 also contains a decremter timer that allows it to enter the nap or doze mode for a specified period and then return to full power operation through the decremter interrupt exception.

The four processor power modes are selectable by setting the appropriate control bits in the `MSR` and `HID0` registers. The following is a brief description of the four power modes:

- Processor full-power—This is the default power state of the MPC8240. The MPC8240 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 enter a low-power state without affecting performance, software execution, or external hardware.
- Processor doze—All the functional units of the processor core are disabled except for the time base/decremter registers and the bus snooping logic. When the processor is in doze mode, an asynchronous interrupt (signalled by the assertion of \overline{int}), a system management interrupt, a decremter exception, a hard or soft reset, or any machine check exception brings the MPC8240 into the full-power state. The MPC8240 in doze mode maintains the PLL in a fully powered state and is locked to the internal `sys_logic_clk` signal so a transition to the full-power state takes only a few processor clock cycles.
- Processor nap—The nap mode further reduces power consumption by disabling bus snooping by the processor, leaving only the time base register and the PLL in a powered state. The MPC8240 returns to the full-power state upon receipt of an interrupt (signalled by the assertion of \overline{int}), a system management interrupt (signalled by the assertion of \overline{SMI}), a decremter exception, a hard or soft reset, or any machine check exception. Also, negation of \overline{QACK} (controlled by the peripheral logic block) causes the processor core to wake up from nap mode. A return to full-power state from a nap state takes only a few processor clock cycles.

- Processor sleep—Sleep mode reduces power consumption to a minimum by disabling all internal functional units, after which external logic can disable the PLL and the internal *sys_logic_clk* signal. The MPC8240 returns to the full-power state from sleep mode upon receipt of an interrupt (signalled by the assertion of \overline{int}), a system management interrupt, a hard or soft reset, or any machine check exception. Also, negation of \overline{QACK} (controlled by the peripheral logic block) causes the processor core to wake up from sleep mode.

The external system logic must enable the processor PLL and the internal *sys_logic_clk* signal before any of the wake-up events occur. Refer to Section 14.3.1.4.2, “Disabling the PLL during Sleep Mode,” for more information on how the PLLs are locked; and Section 3.3, “Clocking,” for more information on the clock signals of the MPC8240.

Note that the processor core cannot switch from one power management mode to another without first returning to full-on mode. Table 14-1 summarizes the four power states for the processor.

Table 14-1. Programmable Processor Power Modes

PM Mode	Functioning Units	Activation Method	Full-Power Wake Up Method
Full power	All units active	—	—
Full power (with DPM)	Requested logic by demand	By instruction dispatch	—
Doze	Bus snooping Data cache as needed Decrementer timer	Controlled by software (write to H1D0)	External asynchronous exceptions (assertion of \overline{SMI} or \overline{int}) Decrementer exception Hard or soft reset Machine check exception (\overline{mcp})
Nap	Decrementer timer	Controlled by software (write to H1D0) and qualified with \overline{QACK} from peripheral logic	External asynchronous exceptions (assertion of \overline{SMI} , or \overline{int}) Decrementer exception Negation of \overline{QACK} by peripheral logic Hard or soft reset Machine check exception (\overline{mcp})
Sleep	None	Controlled by software (write to H1D0) and qualified with \overline{QACK} from peripheral logic	External asynchronous exceptions (assertion of \overline{SMI} , or \overline{int}) Negation of \overline{QACK} by peripheral logic Hard or soft reset Machine check exception (\overline{mcp})

14.2.3 Processor Power Management Modes—Details

The following sections describe the characteristics of the MPC8240 power management modes, the requirements for entering and exiting the various modes, and the system capabilities provided by the processor core while the power management modes are active.

For the processor to enter nap or sleep mode, the software first programs the processor for one of those modes. Then the peripheral logic must be programmed for nap or sleep mode. When the peripheral logic is ready to nap or sleep, it signals that the processor can enter the nap or sleep mode and asserts the \overline{QACK} output signal. If the peripheral logic wakes from nap or sleep (causing \overline{QACK} to negate), the processor also wakes from nap or sleep mode.

14.2.3.1 Full-Power Mode with DPM Disabled

Full-power mode with DPM disabled power mode is selected when the DPM enable bit (bit 11) in $HID0$ is cleared. The following characteristics apply:

- Default state following assertion of $\overline{HRST_CPU}$ and $\overline{HRST_CTRL}$
- All functional units are operating at full processor speed at all times

14.2.3.2 Full-Power Mode with DPM Enabled

Full-power mode with DPM enabled ($HID0[11] = 1$) provides on-chip power management without affecting the functionality or performance of the MPC8240 as follows:

- Required functional units are operating at full processor speed
- Functional units are clocked only when needed
- No software or hardware intervention required after mode is set
- Software/hardware and performance transparent

14.2.3.3 Processor Doze Mode

Doze mode disables most functional units but maintains cache coherency by enabling bus snooping. A snoop hit causes the processor core to enable the data cache, copy the data back to memory, disable the cache, and fully return to the doze state.

Doze mode is characterized by the following features:

- Most functional units disabled
- Bus snooping and time base/decrementer still enabled
- PLL running and locked to the internal sys_logic_clk signal

To enter the doze mode, the following conditions must occur:

- Set doze bit ($HID0[8] = 1$).
- The MPC8240 enters doze mode after several processor clocks.

To return to full-power mode the following conditions must occur:

- Internal \overline{int} or \overline{mcp} signals asserted to the processor by the peripheral logic block, NMI asserted, \overline{SMI} asserted, or decrementer exception
- Hard reset or soft reset
- Transition to full-power state occurs only after a few processor cycles.

14.2.3.4 Processor Nap Mode

The nap mode disables the processor core except for the processor phase-locked loop (PLL) and the time base/decrementer. The time base can be used to restore the processor core to full-on state after a specified period.

Because bus snooping is disabled for nap and sleep mode, the peripheral logic block delays the processor from entering into nap mode until all the peripheral logic internal buffers are flushed and there is no outstanding transaction. Also, software must ensure that no PCI master accesses main memory, unless software can guarantee that none of the accesses would correspond to a modified line in the L1 cache.

When the peripheral logic block has ensured that snooping is no longer necessary, it allows the processor to enter the nap (or sleep) mode and it causes the assertion of the MPC8240 \overline{QACK} output signal for the duration of the nap mode period.

Nap mode is characterized by the following features:

- Time base/decrementer still enabled
- Most functional units disabled (including bus snooping)
- PLL running and locked to the internal *sys_logic_clk* signal

To enter the nap mode, the following conditions must occur:

- Set nap bit ($HID0[9] = 1$).
- Processor asserts internal request for nap or sleep mode to the peripheral logic.
- Peripheral logic must be programmed for nap or sleep mode.
- MPC8240 asserts quiesce acknowledge (\overline{QACK}) output signal after flushing the internal buffers in peripheral logic block.
- Processor core enters nap mode after several processor clocks. (Peripheral logic also enters the nap or sleep mode.)

To return to full-power mode the following conditions must occur:

- Internal \overline{int} or \overline{mcp} signals asserted or \overline{QACK} negated by the peripheral logic block, \overline{SMI} asserted, or decrementer exception
- Hard reset or soft reset
- Transition to full-power takes only a few processor cycles.

14.2.3.5 Processor Sleep Mode

Sleep mode consumes the least amount of power of the four modes since all functional units are disabled. To conserve the maximum amount of power, the processor PLL and the internal *sys_logic_clk* signals can be disabled to the processor. Due to the fully static design of the MPC8240, internal processor state is preserved when no internal clock is present.

Because the time base and decremter are disabled while the processor is in sleep mode, the time base contents must be updated from an external time base following sleep mode if accurate time-of-day maintenance is required.

As in nap mode, the peripheral logic block delays the processor from entering into sleep mode until all the internal buffers are flushed and there is no outstanding transaction. When the peripheral logic block has ensured that snooping is no longer necessary, it allows the processor core to enter the sleep mode and it asserts the \overline{QACK} output signal for the duration of the sleep mode period.

Sleep mode is characterized by the following features:

- All processor functional units disabled (including bus snooping and time base)
- All nonessential input receivers disabled
- Internal clock regenerators disabled
- Processor PLL and the internal *sys_logic_clk* signal can be disabled.

To enter sleep mode the following conditions must occur:

- Set sleep bit ($HID0[10] = 1$).
- Processor asserts internal request for nap or sleep mode to the peripheral logic.
- Peripheral logic must be programmed for nap or sleep mode.
- MPC8240 asserts quiesce acknowledge (\overline{QACK}) output signal after flushing the internal buffers of peripheral logic block.
- Processor core enters sleep mode after several processor clocks. (Peripheral logic also enters the nap or sleep mode.)

To return to full-power mode the following conditions must occur:

- Internal \overline{int} or \overline{mcp} signals asserted or \overline{QACK} negated by the peripheral logic block, NMI asserted, or \overline{SMI} asserted
- Hard reset or soft reset

14.2.4 Power Management Software Considerations

Because the processor core is a dual-issue processor with out-of-order execution capability, care must be taken in how the processor power management modes are entered. Furthermore, nap and sleep modes require all outstanding bus operations to be completed before the processor power management mode is entered. Section 14.4, “Example Code Sequence for Entering Processor Sleep Mode,” provides an example software sequence for putting the processor core into sleep mode.

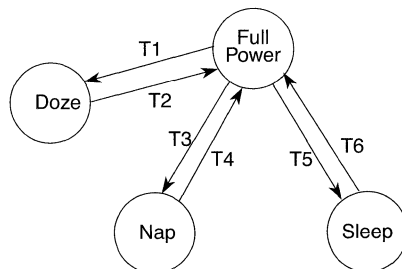
14.3 Peripheral Logic Power Management

Similar to the power management features of the processor core, the peripheral logic block of the MPC8240 has its own doze, nap and sleep modes. They are described in the following subsections.

14.3.1 Peripheral Logic Power Modes

The four power states for the peripheral logic block (three power saving modes plus full-power) of the MPC8240 peripheral logic block are shown in Figure 14-1. The three power saving modes provide different levels of power savings. Doze, nap, and sleep are entered through software by setting the corresponding configuration register bit in the power management control register (PMCR). In addition, the PMCR[PM] global power management bit, must be set to 1 to enable power management of the peripheral logic block.

For the peripheral logic to enter into either nap or sleep mode, the processor must have requested to enter either nap or sleep mode. If the processor wakes up from nap or sleep, the peripheral logic also wakes up from nap or sleep.



T1: PMCR(Doze) = 1 & PMCR(PM) = 1
 T2: hard reset, br = 0, PCI address hit, NMI
 T3: PMCR(Nap) = 1 & PMCR(PM) = 1 & [proc_NAP (or SLEEP) request]
 T4: hard reset, br = 0, PCI address hit, NMI
 T5: PMCR(Sleep) = 1 & PMCR(PM) = 1 & [proc_NAP (or SLEEP) request]
 T6: hard reset, br = 0, NMI

Figure 14-1. MPC8240 Peripheral Logic Power States

In doze, nap, and sleep modes, the peripheral logic always monitors the internal bus request signal from the processor core. When it is asserted, for example due to the occurrence of the decremter exception, the peripheral logic exits its low power state and goes back to the full-power state to service the request.

Note that the MPC8240 does not support the broadcast of PCI special cycles related to low-power operation.

Table 14-2 summarizes the functionality of the peripheral logic block in any power state.

Table 14-2. Programmable Peripheral Logic Power Modes

PM Mode	Functioning Units	Activation Method	Full-Power Wake Up Method
Full power	All units active	—	—
Doze	PCI address decoding and bus arbiter System RAM refreshing Processor bus request and NMI monitoring EPIC unit I ² C unit PLL	Controlled by software (write to PMCR)	PCI access to memory Processor bus request Assertion of NMI Interrupt to EPIC Hard Reset
Nap	PCI address decoding and bus arbiter System RAM refreshing Processor bus request and NMI monitoring EPIC unit I ² C unit PLL	Controlled by software (write to PMCR) and processor core in nap or sleep mode	PCI access to memory ¹ Processor bus request Assertion of NMI Interrupt to EPIC Hard Reset
Sleep	PCI bus arbiter System RAM refreshing (can be disabled) Processor bus request and NMI monitoring EPIC unit I ² C unit PLL (can be disabled)	Controlled by software (write to PMCR) and processor core in nap or sleep mode	Processor bus request Assertion of NMI Interrupt to EPIC Hard Reset

¹A PCI access to memory in nap mode does not cause \overline{QACK} to negate; subsequently, it does not wake up the processor core. Thus, the processor won't snoop this access. After servicing the PCI access, the peripheral logic automatically returns to the nap mode.

14.3.1.1 Peripheral Logic Full Power Mode

The default power state of the MPC8240 is full-power. In this state, the peripheral logic block is fully powered, and the internal functional units are operating at full clock speed.

14.3.1.2 Peripheral Logic Doze Mode

In this power saving mode, all functional units of the peripheral logic block are disabled except for PCI address decoding, the PCI bus arbiter, system RAM refreshing, processor bus request monitoring, and NMI signal monitoring. Also, the EPIC and I²C units continue to function.

When the doze state is entered, a PCI transaction referenced to the system memory, a processor bus request assertion of NMI (provided PICR1[MCP_EN] = 1), assertion of \overline{int} from the EPIC unit, or a hard reset brings the peripheral logic out of the doze mode and into the full-power state. Note that other processor exceptions (for example, assertion of the \overline{SMI} signal) cause processor bus cycles; therefore, they indirectly cause the peripheral logic to wake up from doze mode.

After the request has been serviced, the peripheral logic goes back to the doze state unless PMCR[DOZE] or PMCR[PM] has been cleared and no pending operations exist.

In doze mode, the PLL is fully operational and locked to PCI_SYNC_IN. The transition to the full power state takes only a few processor clock cycles. The peripheral logic doze mode operates totally independently from the power saving state of the processor.

14.3.1.3 Peripheral Logic Nap Mode

Further power saving from doze mode can be guaranteed through the peripheral logic nap mode because the peripheral logic does not enter the nap mode unless the processor core is in either nap or sleep mode.

As in peripheral logic doze mode, all peripheral logic functional units are disabled except for PCI address decoding, the PCI bus arbiter, system RAM refreshing, processor bus request monitoring, and NMI signal monitoring. Also, the EPIC and I²C units continue to function.

When the nap mode is entered, a PCI transaction referenced to the system memory, a processor bus request the assertion of NMI (PICR1[MCP_EN] must be set to recognize NMI), or a hard reset brings the peripheral logic out of the nap mode and into the full-power state.

14.3.1.3.1 PCI Transactions During Nap Mode

When the peripheral logic is awakened from the nap state by a PCI-agent initiated transaction, the transaction is serviced, PMCR[PM] remains set, \overline{QACK} does not negate (so the processor core does not wake from nap or sleep), and the peripheral logic goes back to the nap state after the transaction has been serviced. If the peripheral logic is awakened by any other cause, PMCR[PM] is cleared, preventing the peripheral logic from automatically re-entering the nap state.

Note that if the MPC8240 is temporarily awakened to service a PCI transaction, the processor core is still in either nap or sleep mode; therefore, it does not respond to any snoop cycles. Software should therefore flush the L1 cache before allowing the peripheral logic to enter the nap mode if the system allows PCI accesses to wake up the peripheral logic without waking the processor core.

14.3.1.3.2 PLL Operation During Nap Mode

In peripheral logic nap mode, the PLL is fully operational and locked to PCI_SYNC_IN. The transition to the full-power state should take no more than a few processor cycles.

14.3.1.4 Peripheral Logic Sleep Mode

Peripheral logic sleep mode provides further power saving compared to the nap mode. Similar to nap mode, the peripheral logic does not enter the sleep mode unless the processor core has requested to enter either nap or sleep mode.

In peripheral logic sleep mode, no functional units are operating except the on-chip PCI bus arbiter, system RAM refreshing logic (enabled by PMCR[LP_REF_EN] for sleep mode), processor bus request monitoring, NMI signal monitoring, the EPIC unit, and the I²C unit. Similar to nap mode, a processor bus request (provided PMCR[CKO_SEL] is set and the

assertion of NMI (PICR1[MCP_EN] is set to recognize NMI) or a hard reset brings the peripheral logic out of the sleep mode and into the full-power state. In addition, external interrupts to the EPIC unit or to the processor core wake the core logic up from the sleep state. PMCR[PM] is always cleared after the core logic is awakened from the sleep state.

14.3.1.4.1 System Memory Refresh during Sleep Mode

In sleep mode, the system memory contents can be maintained either by enabling the memory's self refresh mode or by having the operating system copy all the memory contents to the hard disk before the peripheral logic enters the sleep state. Alternatively, the MCP8240 refresh logic continues to perform the refresh function in sleep mode if PMCR[LP_REF_EN] is set. In this case, MCCR1[SREN] is used to determine whether the refresh is a self refresh (SREN = 1) or a CBR refresh (SREN = 0). If LP_REF_EN is cleared, the refresh operations stop when the MPC8240 enters the sleep mode.

When the core logic is in the sleep state, using CBR refresh, and keeping the PLL in locked operation, the wake up latency is comparable to that of nap mode (a few processor clocks). However, additional wake up latency is incurred if the system uses the self refresh mode and/or turns off the PLL during the sleep state.

14.3.1.4.2 Disabling the PLL during Sleep Mode

In the peripheral logic sleep mode, the PLL for the peripheral logic block and the PCI_SYNC_IN input may be disabled by an external power management controller (PMC) for further power saving. The PLL can be disabled by setting the PLL_CFG(0–4) pins to the PLL bypass mode. Disabling the PLL and/or the PCI_SYNC_IN input during sleep mode should not occur until after the assertion of \overline{QACK} , ensuring that the processor is in either nap or sleep mode.

If the peripheral logic PLL is disabled, the external PMC chip should trap all the wake up events so that it can turn on the PLL (to guarantee the relock time) and/or the PCI_SYNC_IN input before forwarding the wake up event to the MPC8240. When recovering from sleep mode, the external PMC has to re-enable the PLL and PCI_SYNC_IN first, and then wake up the MPC8240 (using any of the wake-up methods) after 100 μ s of PLL relock time.

14.3.1.4.3 PCI Transactions in Sleep Mode

PCI transactions are not serviced in peripheral logic sleep mode. It is important to guarantee that no new PCI transactions will occur before the PMCR is programmed for sleep mode.

14.3.1.4.4 SDRAM Paging During Sleep Mode

SDRAM systems that have paging mode enabled must disable paging mode before entering the sleep mode, to avoid memory loss and corruption. After the peripheral logic exits the sleep mode and returns to the full-power state, paging mode can once again be enabled.

14.4 Example Code Sequence for Entering Processor Sleep Mode

The following is a sample code sequence for entering processor core sleep mode with the MPC8240.

```

*****
# first set up MPC8240 Power Management Register
# and the processor core HID0 power management bits
*****
# turn on power management bits
#
    addis    r3, r0, 0x8000    # start building new register number
    ori     r3, r3, 0x0070    # register number 0xf0
    stwbrx  r3, r0, r1        # write this value to CONFIG_ADDR
    sync
    lhbrx   r4, r0, r2        # load r4 from CONFIG_DATA
    addis   r0, r0, 0x0000    # PM=1
    ori     r0, r0, 0xc088    # set bits 15, 14, and 7, 3(sleep)
    or      r4, r4, r0        # set the PM bit
    sync
    sthbrx  r4, r0, r2        # write the modified data to
CONFIG_DATA
    sync

*****processor HID and external interrupt initialization*****
#
# set up HID registers for the various PowerPC processors
# hid setup taken from minix's mpxPowerPC.s

    mfspr   r31, pvr          # pvr reg
    srawi   r31, r31, 16
resetTest603:
    cmpi    0, 0, r31, 3
    bne     cr0, endHIDSetup

    addi    r0, r0, 0
    oris   r0, r0, 0x1000    # enable machine check pin EMCP
    oris   r0, r0, 0x0010    # enable dynamic power mgmt DPM
    oris   r0, r0, 0x0020    # enable SLEEP power mode
    ori    r0, r0, 0x8000    # enable the Icache ICE
    ori    r0, r0, 0x4000    # enable the Dcache DCE
    ori    r0, r0, 0x0800    # invalidate Icache ICFI
    ori    r0, r0, 0x0400    # invalidate Dcache DCFI
    mtspr  hid0, r0
    isync

*****
# then when the processor is in a loop, force an SMI interrupt
*****

.orig 0x00001400                # System Management Interrupt

# force big endian mode
    stw    r0,0x05f8,r0        # need nop every second inst.
    stw    r0,0x05fc,r0
    mfmsr  r0
    ori    r0,r0,r0
    ori    r0,r0,0x0001        # force big endian LE bit
    ori    r0,r0,r0
    xori   r0,r0,0x0001        # force big endian LE bit

```

Example Code Sequence for Entering Processor Sleep Mode

```

ori      r0,r0,r0
mtmsr   r0
ori      r0,r0,r0
isync
ori      r0,r0,r0

# save off additional registers to be corrupted
stw     r20,0x05f4,r0
mfspr   r21, srr0      # put srr0 in r21
stw     r21,0x05f0,r0  # put r21 in 0x05f0
mfspr   r22, srr1      # put srr1 in r22
stw     r22,0x05ec,r0  # put r22 in 0x05ec
stw     r23,0x05e8,r0
mfcrr   r23
stw     r23,0x05e4,r0
xor     r0,r0,r0

*****
# set msr pow bit to go into sleep mode

sync
mfmsr   r5             # get MSR
addis   r3, r0, 0x0004 # turn on POW bit
ori     r3, r3, 0x0000 # turn on ME bit 19
or     r5, r3, r5
mtmsr   r5
isync

addis   r20, r0, 0x0000
ori     r20, r20, 0x0002
stay_here:
addic.  r20, r20, -1   # subtract 1 from r20 and set cc
bgt     cr0, stay_here # loop if positive

# restore corrupted registers
lwz     r23,0x05e4,r0
mtcrf   0xff,r23
lwz     r23,0x05e8,r0
lwz     r22,0x05ec,r0
mfspr   srr1, r22
lwz     r21,0x05f0,r0
mfspr   srr0, r21
lwz     r20,0x05f4,r0
lwz     r0,0x05fc,r0

sync
rfi

*****
# to get out of sleep mode, do a Soft Reset
*****

.orig 0x00000100          # Reset handler in low memory

# force big endian mode
stw     r0,0x05f8,r0      # need nop every second inst.
stw     r0,0x05fc,r0
mfmsr   r0
ori     r0,r0,r0
ori     r0,r0,0x0001      # force big endian LE bit
ori     r0,r0,r0
xori    r0,r0,0x0001      # force big endian LE bit
ori     r0,r0,r0
mtmsr   r0

```


Example Code Sequence for Entering Processor Sleep Mode

```
    ori        r0,r0,r0
    isync
    ori        r0,r0,r0

# save off additional registers to be corrupted
    stw        r20,0x05f4,r0
    stw        r21,0x05f0,r0
    stw        r22,0x05ec,r0
    stw        r23,0x05e8,r0
    mfcrr     r23
    stw        r23,0x05e4,r0
    xor        r0,r0,r0

# restore corrupted registers
    lwz        r23,0x05e4,r0
    mtcrf     0xff,r23
    lwz        r23,0x05e8,r0
    lwz        r22,0x05ec,r0
    lwz        r21,0x05f0,r0
    lwz        r20,0x05f4,r0
    lwz        r0,0x05fc,r0

    sync
    rfi
#*****
```

Example Code Sequence for Entering Processor Sleep Mode

Chapter 15

Debug Features

The MPC8240 provides several features to aid in the process of system bring-up and debug. This chapter discusses the following topics:

- Address attributes signals—Identification of the source and type of transaction currently being performed on either the PCI or memory interface.
- Debug address signals—Supplementation of the DRAM column address and chip select signals, allowing the system to reconstruct the corresponding physical address on the internal peripheral logic bus in a single cycle.
- $\overline{\text{MIV}}$ signal—Marking of valid address and data bus cycles on the memory bus.
- Memory data path error injection/capture—Injection of multi-bit stuck-at faults onto the peripheral logic or memory data/parity buses and capture the data/parity upon the receipt of an ECC or parity error.
- JTAG/testing support

15.1 Address Attribute Signals

The MPC8240 provides additional information corresponding to memory and PCI activity on several signals in order to assist with system debug. These signals are the memory attribute signals (MAA[0–2]) and the PCI attribute signals (PMAA[0–2]) as summarized in Table 15-1.

Table 15-1. Address Attribute Signal Summary

Signal Name	Pins	I/O	Signal Meaning
MAA[0–2]	3	O	Memory address attributes
PMAA[0–2]	3	O	PCI address attributes

15.1.1 Memory Address Attribute Signals (MAA[0–2])

The memory attribute signals are associated with the memory interface and provide information about the source of the memory operation being performed by the MPC8240.

The encodings of the memory address attribute signals are defined in Table 15-2.

Table 15-2. Memory Address Attribute Signal Encodings

MAA0	MAA1	MAA2	Memory Operation	Definition
0	0	0	read	Processor data read
0	0	1	read	Processor touch load
0	1	0	read	Processor instruction fetch
0	1	1	—	reserved
1	0	0	read	PCI memory read
1	0	1	read	DMA channel 0 memory read
1	1	0	read	DMA channel 1 memory read
1	1	1	read	I ² O memory read
0	0	0	write	Processor data write
0	0	1	—	reserved
0	1	0	—	reserved
0	1	1	—	reserved
1	0	0	write	PCI memory write
1	0	1	write	DMA channel 0 memory write
1	1	0	write	DMA channel 1 memory write
1	1	1	write	I ² O memory write

15.1.2 Memory Address Attribute Signal Timing

The memory address attribute signals have timing characteristics as shown in Figure 15-8 through Figure 15-16. These figures also show the relationship of MAA[0–2] with the \overline{MIV} signal. Note that the attribute signals are valid at the same time as the column address for all DRAM accesses including single-beat, burst, 32- and 64-bit modes, page misses, page hits and others. Note that for all ROM/Flash accesses the attribute signals are valid when the address is valid.

15.1.3 PCI Address Attribute Signals

The PCI address attribute signals provide information about the source of the PCI operation being performed by MPC8240, and the encodings are defined in Table 15-3.

Table 15-3. PCI Attribute Signal Encodings

PMAA0	PMAA1	PMAA2	PCI Operation	Definition
0	0	0	read	Processor data load

Table 15-3. PCI Attribute Signal Encodings

0	0	1	read	Processor touch load
0	1	0	read	Processor instruction fetch
0	1	1	—	reserved
1	0	0	—	reserved
1	0	1	read	DMA channel 0 PCI read
1	1	0	read	DMA channel 1 PCI read
1	1	1	read	PCI address bus invalid
0	0	0	write	Processor data write
0	0	1	—	reserved
0	1	0	—	reserved
0	1	1	—	reserved
1	0	0	—	reserved
1	0	1	write	DMA channel 0 PCI write
1	1	0	write	DMA channel 1 PCI write
1	1	1	write	PCI address bus invalid

15.1.4 PCI Address Attribute Signal Timing

The PCI attribute signals have timing characteristics as shown in Figure 15-1 and Figure 15-2. Note that the attribute signals are valid at the same time as the address for all MPC8240-sourced PCI accesses. During all other clock cycles, PMAA[0–2] are held at the value 0b111.

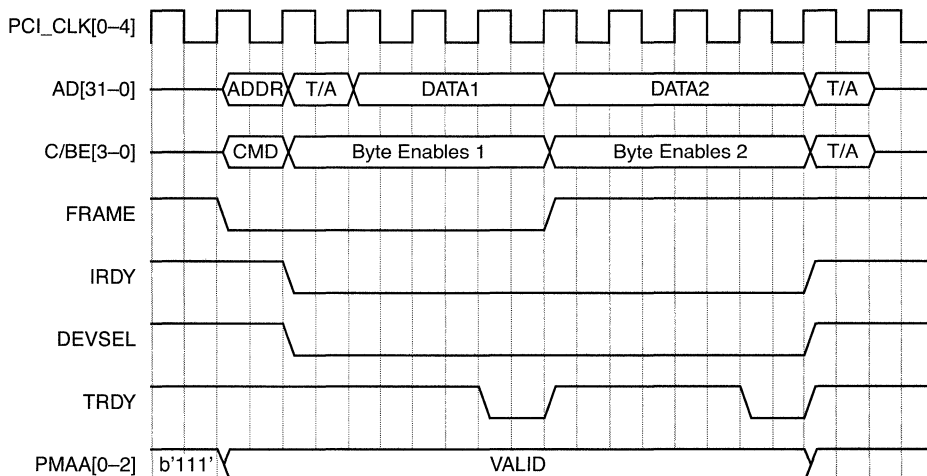


Figure 15-1. Example PCI Address Attribute Signal Timing for Burst Read Operations

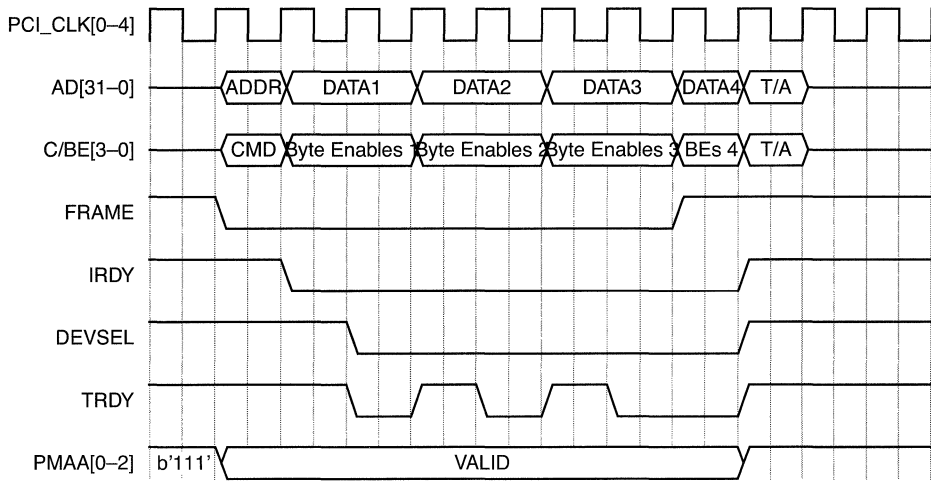


Figure 15-2. Example PCI Address Attribute Signal Timing for Burst Write Operations

15.2 Memory Debug Address

When enabled, the debug address gives software disassemblers a simple way to reconstruct the 30-bit physical address for a memory bus transaction to DRAM, SDRAM, ROM, Flash, or Port X. For DRAM or SDRAM, these 16 debug address signals are sampled with the column address and chip-selects. For ROM, Flash, and Port X devices, the debug address pins are sampled at the same time as the ROM address and can be used to recreate the 24-bit physical address in conjunction with ROM address. The granularity of the reconstructed physical address is limited by the bus width of the interface; double-words for 64-bit interfaces, words for 32-bit interfaces, and bytes for 8-bit interfaces.

15.2.1 Enabling Debug Address

The debug address functionality is enabled or disabled at reset by using the $\overline{\text{GNT4}}$ reset configuration signal. If the $\overline{\text{GNT4}}$ signal is left floating at reset, an internal pull-up forces it high, disabling the debug address functionality. If $\overline{\text{GNT4}}$ is asserted at reset (driven low), the debug address functionality is enabled. See Section 3.4, “Configuration Pins Sampled at Reset,” for a complete description of all the reset configuration signals.

15.2.2 Debug Address Signal Definitions

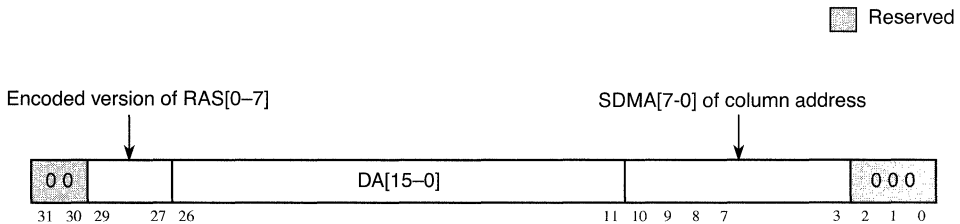
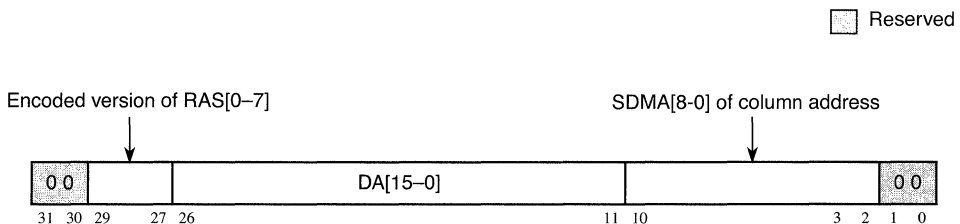
Table 15-4 describes the mapping of all the memory debug address signals and their alternate functions. Note that while DA[15–0] are all outputs when used as the debug address signals, the alternate functions for some of these signals are defined as inputs.

Table 15-4. Memory Debug Address Signal Definitions

Signal	Signal Meaning	Alternate Function	Pins	I/O
DA[15–11]	debug_address[15–11]	—	5	O
DA[10–6]	debug_address[10–6]	PLL_CFG[0–4]	5	O
DA5	debug_address 5	$\overline{\text{GNT4}}$	1	O
DA4	debug_address 4	REQ4	1	O
DA3	debug_address 3	PCI_CLK4	1	O
DA2	debug_address 2	—	1	O
DA1	debug_address 1	CKO	1	O
DA0	debug_address 0	$\overline{\text{QACK}}$	1	O
$\overline{\text{GNT4}}$	Debug address enable (during reset configuration). 0 Debug address enabled; partial address of the transaction driven on DA[15-0]. 1 Debug address disabled	$\overline{\text{GNT4}}$; DA5	1	I

15.2.3 Physical Address Mappings

The physical address mappings for 64- and 32-bit DRAM and SDRAM are depicted in Figure 15-3 and Figure 15-4. The physical address mappings for 64-, 32-, and 8-bit ROM/Flash are depicted in Figure 15-5, Figure 15-6, and Figure 15-7 respectively.

**Figure 15-3. 64-Bit Mode, DRAM and SDRAM Physical Address for Debug****Figure 15-4. 32-Bit Mode, DRAM and SDRAM Physical Address for Debug**

Reserved

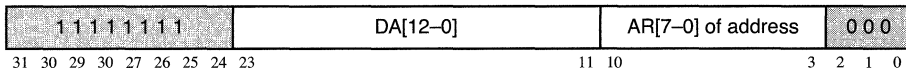


Figure 15-5. 64-Bit Mode, ROM and Flash Physical Address for Debug

Reserved

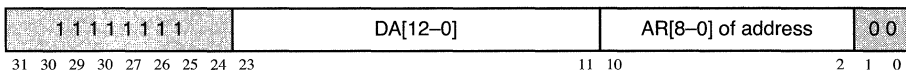


Figure 15-6. 32-Bit Mode, ROM and Flash Physical Address for Debug

Reserved

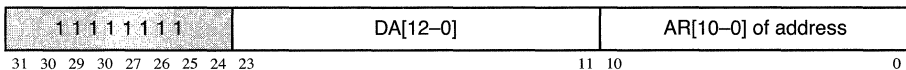


Figure 15-7. 8-bit mode, ROM and FLASH Physical Address for Debug

15.2.4 RAS/CS Encoding

The encoding of $\overline{\text{RAS/CS}}[0-7]$ to form bits 29–27 of the physical address for DRAM and SDRAM transactions is based on the memory bank configuration as programmed in the bank starting and ending address configuration registers; registers 0x80, 0x84, 0x88, 0x8C, 0x90, 0x94, 0x98, and 0x9C. For this encoding algorithm to be deterministic, DRAM and SDRAM banks are not allowed to cross a 128 Mbyte address partition (that is, the starting and ending address for any one bank must fall within the same 128 Mbyte partition). Obviously, such $\overline{\text{RAS/CS}}$ information is relevant only for DRAM (and SDRAM) and not for ROM/Flash. For a simple example of RAS encodings, see Table 15-5 below.

Table 15-5. Example of Chip Select Encoding For 568 Mbyte Memory System

Partition	Encoded RAS/CS: Physical Address[29–27]	Address		Bank	Bank Size	$\overline{\text{RAS}}[0-7]$
		Ending	Starting			
Eighth 128 Mbyte partition of 1 Gbyte main memory	b'111'	Ending	0x3FF	undefined		
		Starting	0x380			

Table 15-5. Example of Chip Select Encoding For 568 Mbyte Memory System

Partition	Encoded RAS/CS: Physical Address[29–27]	Address		Bank	Bank Size	$\overline{\text{RAS}}[0-7]$
		Ending	Starting			
Seventh 128 Mbyte partition of 1 Gbyte main memory	b'110'	Ending	0x37F	undefined		
		Starting	0x300			
Sixth 128 Mbyte partition of 1 Gbyte main memory	b'101'	Ending	0x2FF	undefined		
		Starting	0x280			
Fifth 128 Mbyte partition of 1 Gbyte main memory	b'100'	Ending	0x27F	undefined		
		Starting	0x238			
		Ending	0x237	7	8 MBytes	0xFE
		Starting	0x230			
		Ending	0x22F	6	16 MBytes	0xFD
		Starting	0x220			
		Ending	0x21F	5	32 MBytes	0xFB
		Starting	0x200			
Fourth 128 Mbyte partition of 1 Gbyte main memory	b'011'	Ending	0x1FF	4	64 MBytes	0xF7
		Starting	0x1C0			
		Ending	0x1BF	3	64 MBytes	0xEF
		Starting	0x180			
Third 128 Mbyte partition of 1 Gbyte main memory	b'010'	Ending	0x17F	1	128 MBytes	0xBF
		Starting	0x100			
Second 128 Mbyte partition of 1 Gbyte main memory	b'001'	Ending	0x0FF	2	128 MBytes	0xDF
		Starting	0x080			
First 128 Mbyte partition of 1 Gbyte main memory	b'000'	Ending	0x07F	0	128 MBytes	0x7F
		Starting	0x000			

15.2.5 Debug Address Timing

For examples of debug address timing for various memory types, see Figure 15-8 through Figure 15-16.

15.3 Memory Interface Valid ($\overline{\text{MIV}}$)

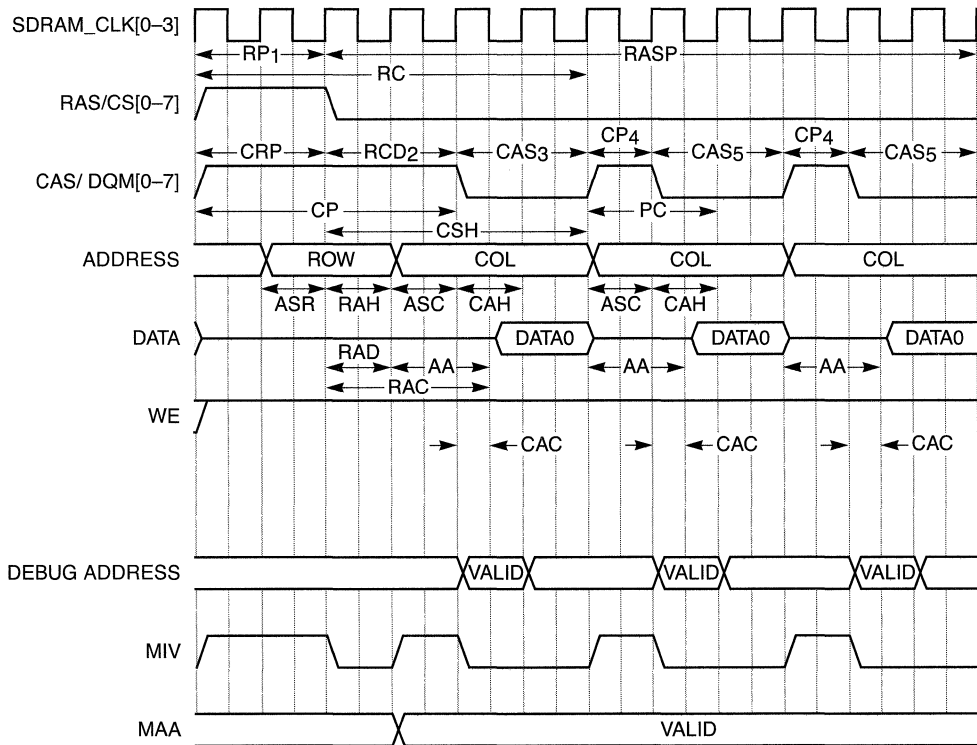
The memory interface valid signal $\overline{\text{MIV}}$ is asserted whenever FPM, EDO, SDRAM, Flash, or ROM addresses or data are present on the external memory bus. It is intended to help reduce the number of bus cycles that logic analyzers must store in memory during a debug trace and is described in Table 15-6. The $\overline{\text{MIV}}$ signal should be sampled with the rising edge of SDRAM_CLK[0–3].

Table 15-6. Memory Interface Valid Signal Definition

Signal Name	Pins	Active	I/O	Signal Meaning
$\overline{\text{MIV}}$	1	Low	O	Indicates that the transaction address or data is valid on the memory bus.

15.3.1 $\overline{\text{MIV}}$ Signal Timing

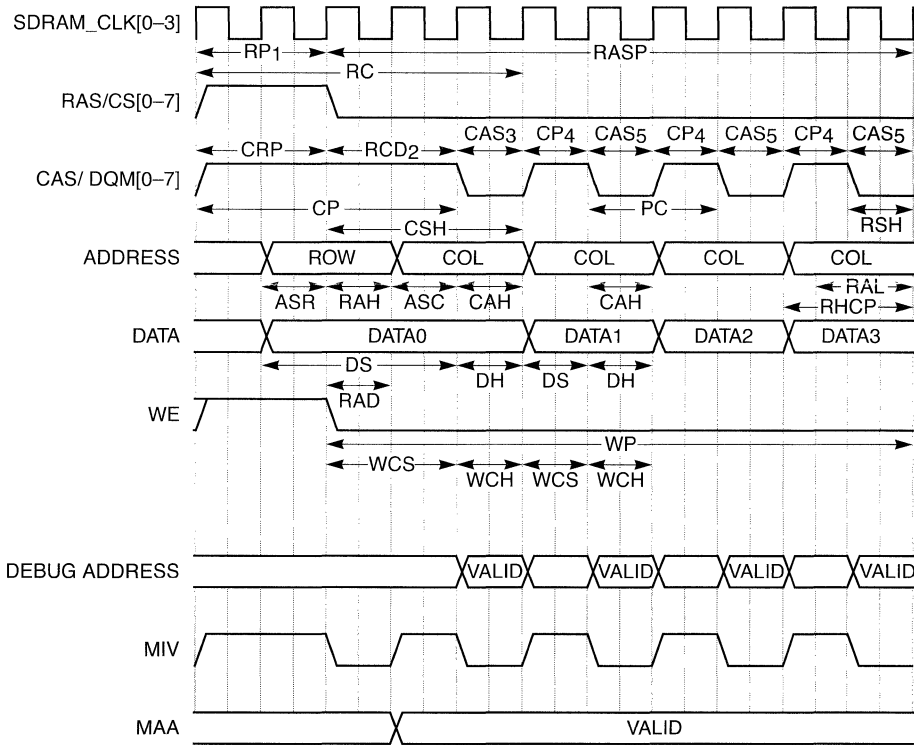
The $\overline{\text{MIV}}$ signal is an active low signal and has timing characteristics as shown in Figure 15-8 through Figure 15-16.



NOTES:

1. Subscripts identify programmable timing variables (RP₁, RCD₂, CAS₃).
2. $\overline{\text{MIV}}$ asserts for address and control on the first clock cycle that RAS or CAS is asserted for a read.
3. $\overline{\text{MIV}}$ asserts for data on the last clock cycle that CAS is asserted for a read.

Figure 15-8. Example FPM Debug Address, $\overline{\text{MIV}}$, and MAA Timings for Burst Read Operation

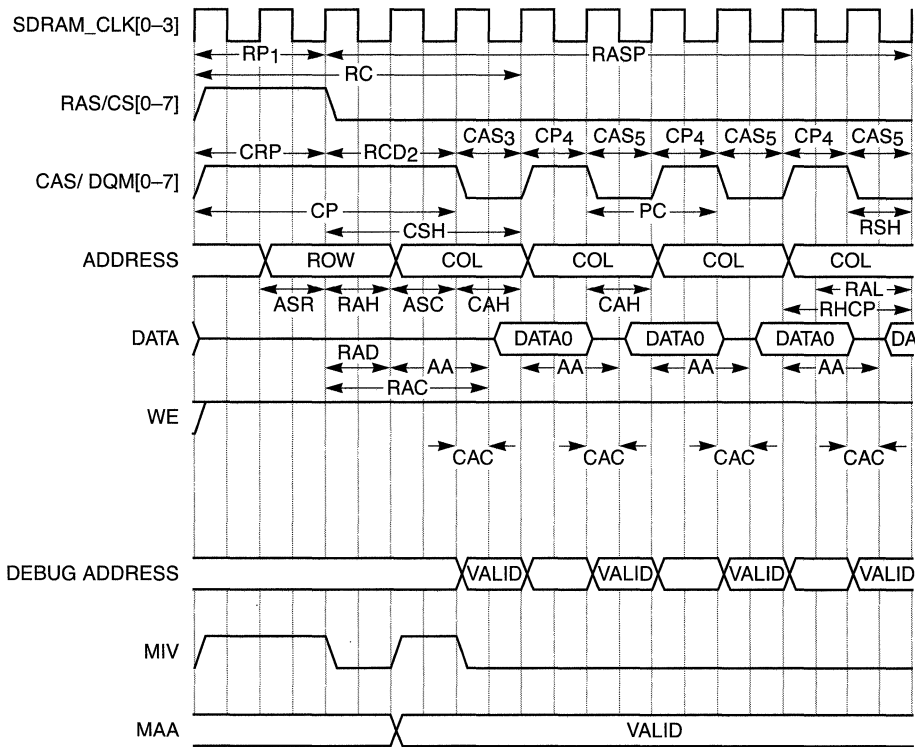


NOTES:

1. Subscripts identify programmable timing variables (RP1, RCD2, CAS3).
2. MIV asserts for address, control, and data on the first clock cycle that RAS or CAS is asserted for a write.

Figure 15-9. Example FPM Debug Address, \overline{MIV} , and MAA Timings for Burst Write Operation

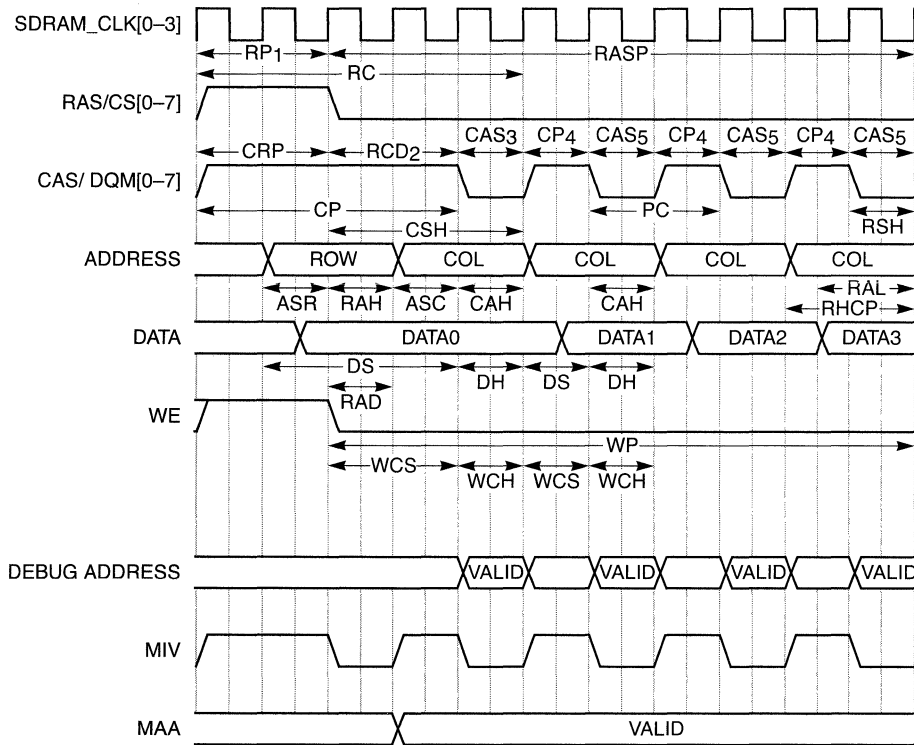
Memory Interface Valid (MIV)



NOTES:

1. Subscripts identify programmable timing variables (RP1, RCD2, CAS3).
2. MIV asserts for address and control on the first clock cycle that RAS or CAS is asserted for a read.
3. MIV asserts for data on the same clock cycle that CAS negates for a read.

Figure 15-10. Example EDO Debug Address, $\overline{\text{MIV}}$, and MAA Timings for Burst Read Operation



NOTES:

1. Subscripts identify programmable timing variables (RP_1 , RCD_2 , CAS_3).
2. MIV asserts for address, control, and data on the first clock cycle that RAS or CAS is asserted for a write.

Figure 15-11. Example EDO Debug Address, \overline{MIV} , and MAA Timings for Burst Write Operation

Memory Interface Valid (MIV)

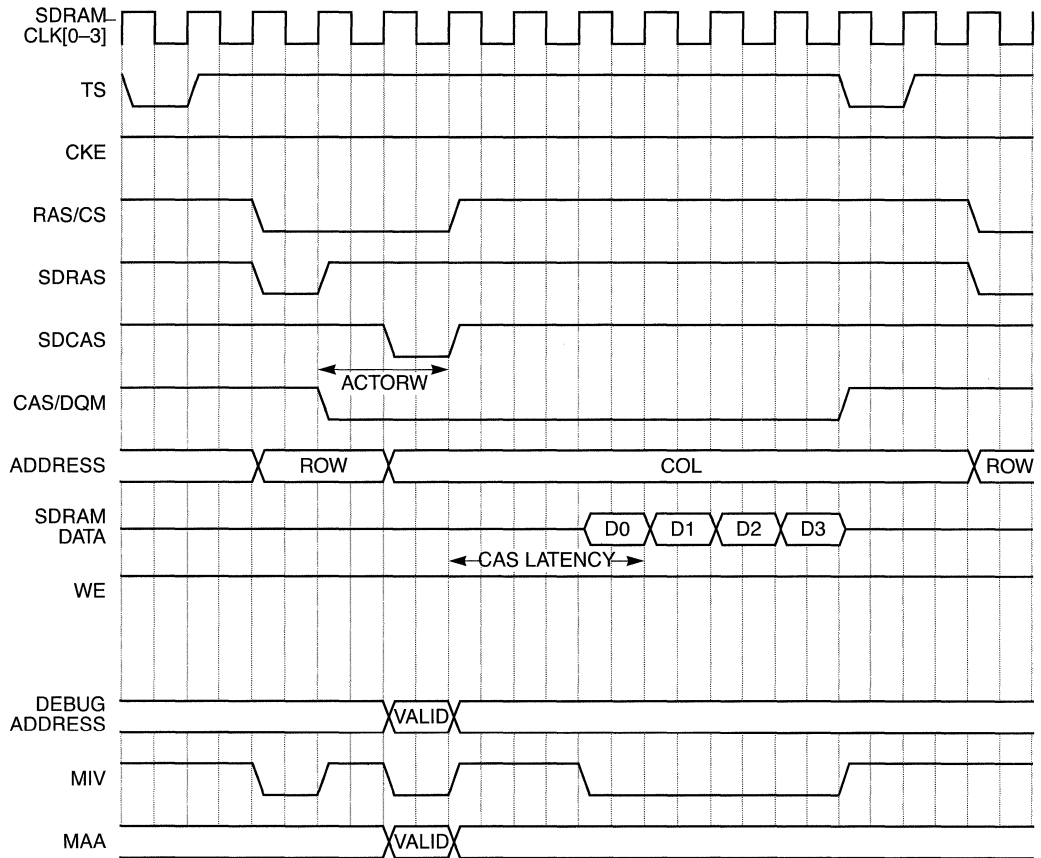


Figure 15-12. Example SDRAM Debug Address, \overline{MIV} , and MAA Timings for Burst Read Operation

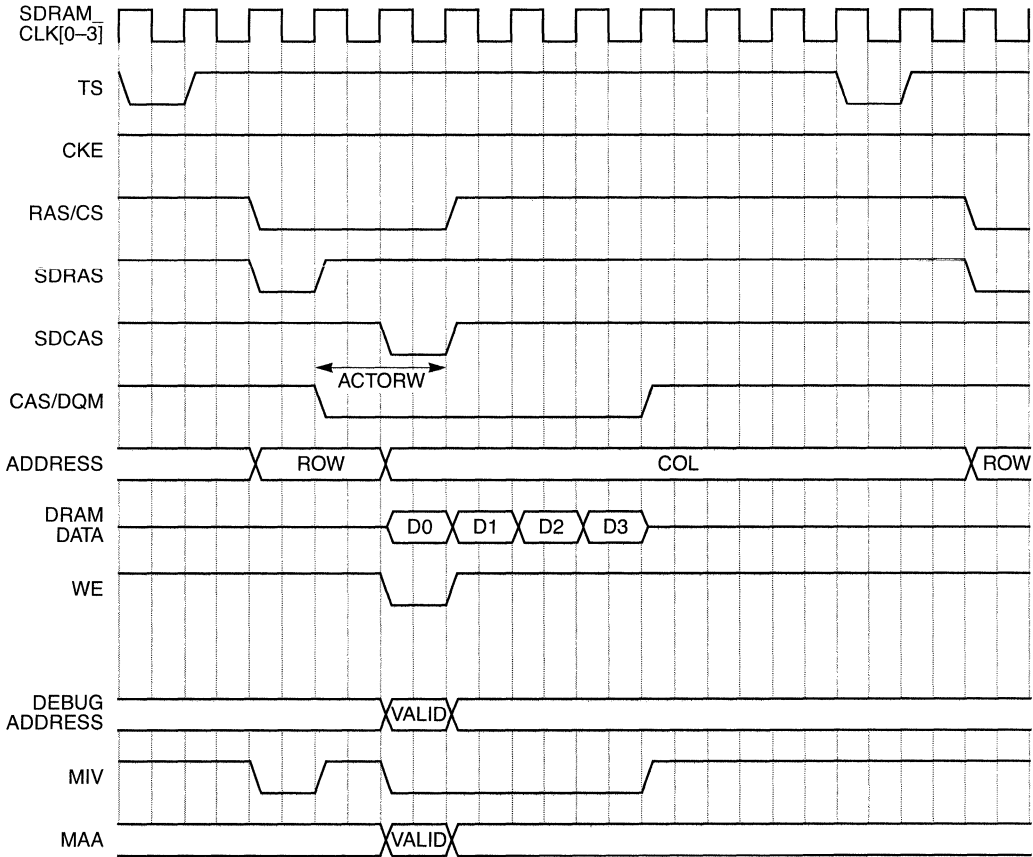
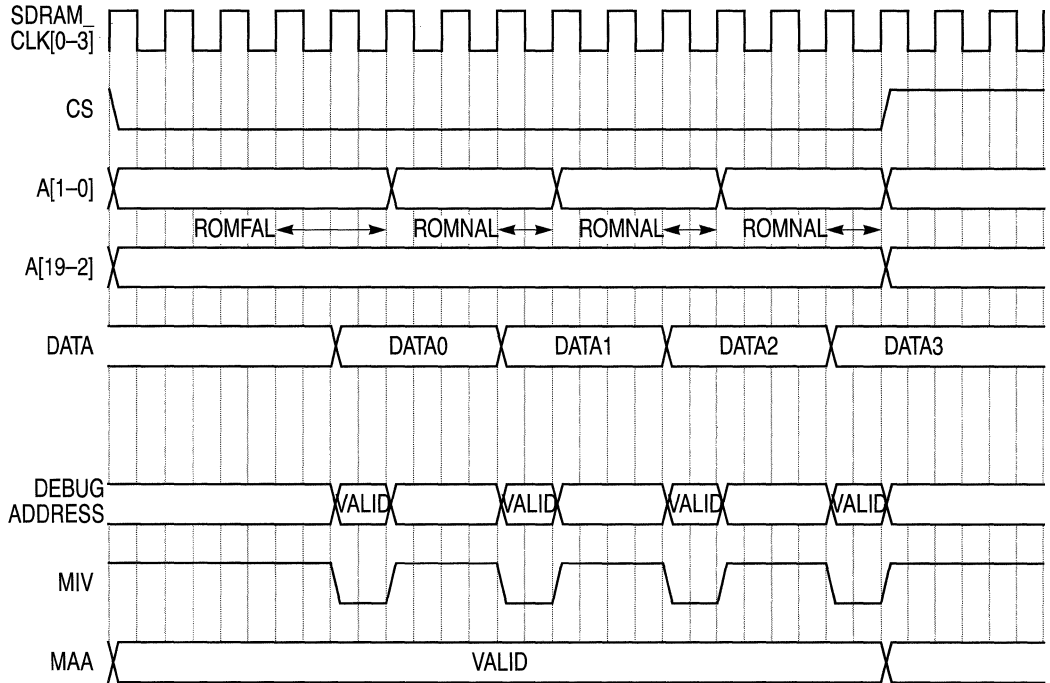


Figure 15-13. Example SDRAM Debug Address, \overline{MIV} , and MAA Timings for Burst Write Operation

Memory Interface Valid (MIV)



NOTES:

1. ROMFAL (ROM First Access Latency) = 0–15 clocks.
2. ROMNAL (ROM Nibble Access Latency) = 0–9 clocks.
3. Memory configuration BURST = 1.

Figure 15-14. Example ROM Debug Address, \overline{MIV} , and MAA Timings For Burst Read

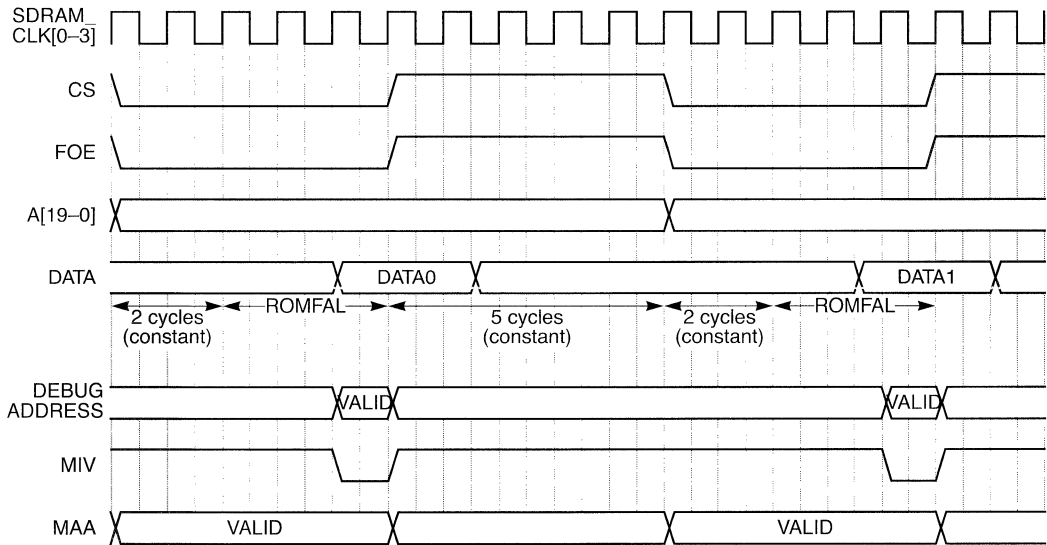


Figure 15-15. Example Flash Debug Address, $\overline{\text{MIV}}$, and MAA Timings For Single-Byte Read

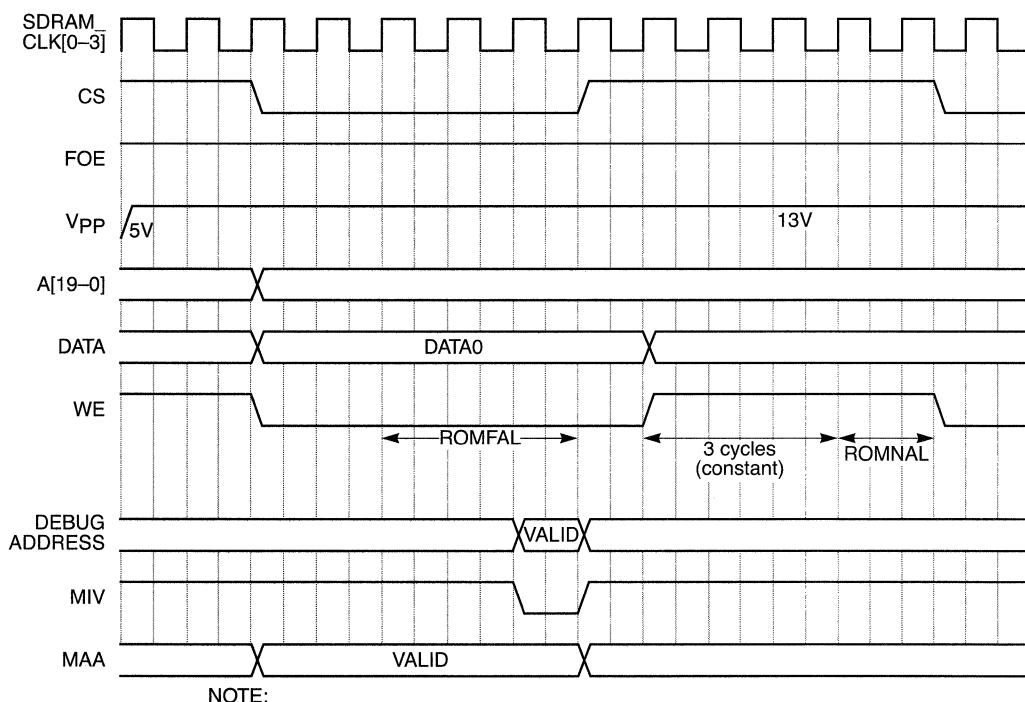


Figure 15-16. Example Flash Debug Address, \overline{MIV} , and MAA Timings for Write Operation

15.4 Memory Datapath Error Injection/Capture

The MPC8240 provides hardware to exercise and debug the on-chip ECC and parity logic by allowing the user to inject multi-bit stuck-at faults onto the peripheral logic or memory data/parity buses and to capture the data/parity upon the receipt of an ECC or parity error.

The memory data path error injection/capture system is programmed via the six memory data path diagnostic registers. These registers allow the user to program error injection masks and to monitor ECC/parity error capture data. The memory data path diagnostic registers are accessible from either the local bus or PCI port. All memory data path diagnostic registers are reset to zero, 0x0000_0000, unless otherwise specified.

15.4.1 Memory Data Path Diagnostic Registers Address Map

The six data path diagnostic registers consist of three error injection mask registers and three error capture monitor registers and they are mapped as follows:

- Embedded utilities memory block (EUMBBAR) for local bus accesses at offsets 0xFF000 to 0xFFFFF.
- Embedded utilities peripheral control and status registers (PCSRBAR) for PCI bus accesses at offsets 0xF00 to 0xFFF.

The offsets to individual registers are defined in Table 15-7.

Table 15-7. Memory Data Path Diagnostic Register Offsets

Local Bus Offset	PCI Bus Offset	Size	Program Access Size	Register	Register Access	Reset Value
0xFF000	0xF00	4-bytes	4 bytes	MDP_ERR_INJ_MASK_DH	Read/Write	0x0000_0000
0xFF004	0xF04	4-bytes	4 bytes	MDP_ERR_INJ_MASK_DL	Read/Write	0x0000_0000
0xFF008	0xF08	4-bytes	1, 2, or 4 bytes	MDP_ERR_INJ_MASK_PAR	Read/Write	0x0000_0000
0xFF00C	0xF0C	4-bytes	4 bytes	MDP_ERR_CAP_MON_DH	Read Only	0x0000_0000
0xFF010	0xF10	4-bytes	4 bytes	MDP_ERR_CAP_MON_DL	Read Only	0x0000_0000
0xFF014	0xF14	4-bytes	1, 2, or 4 bytes	MDP_ERR_CAP_MON_PAR	Read/Write	0x0000_0000

15.4.2 Memory Data Path Error Injection Mask Registers

The memory data path error injection masks are used to inject errors onto the internal peripheral logic bus or the memory data/parity buses as shown in Figure 15-17. Separate mask registers are provided for the high data, low data, and parity buses. The masks are bit-wise inverting; a 0b1 in the mask causes the corresponding bit on the data/parity bus to be inverted. The masks are applied to the read and/or write data path by read and write mask enable bits. These registers can be read or written and are initialized to 0x0000_0000.

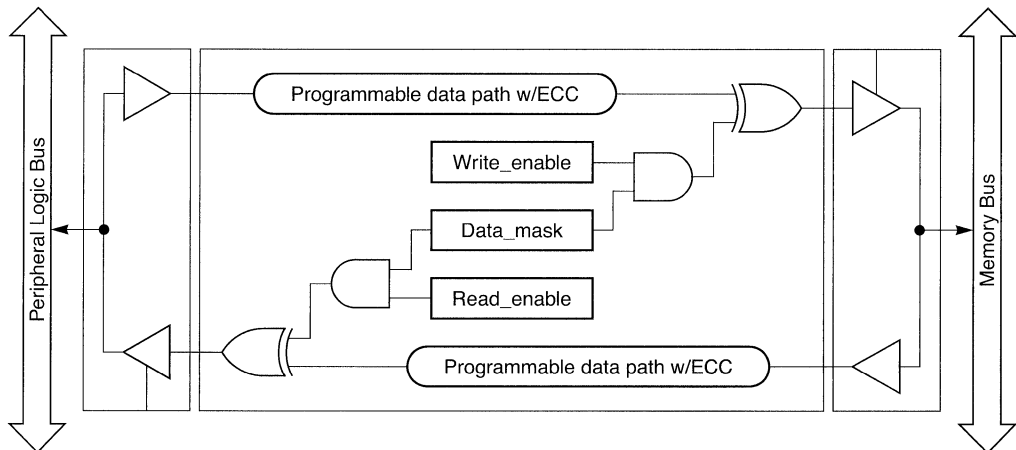



Figure 15-17. Functional Diagram of Memory Data Path Error Injection

15.4.2.1 Data High Error Injection Mask Register

Figure 15-18 shows the bits of the data high error injection mask register.

 Reserved

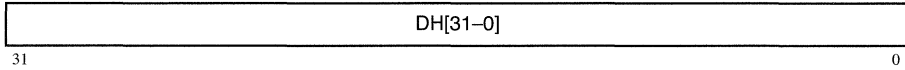


Figure 15-18. Data High Error Injection Mask (MDP_ERR_INJ_MASK_DH): 4Bytes @ <FF000, F00>

Figure 15-8 shows the bit definitions of the data high error injection mask register.

Table 15-8. Data High Error Injection Mask Bit Field Definitions

Bit	Name	Reset Value	R/W	Description
31-0	DH[31-0]	all 0s	R/W	Error injection mask for memory data path data bus high

15.4.2.2 Data Low Error Injection Mask Register

Figure 15-19 shows the bits of the data low error injection mask register.

 Reserved

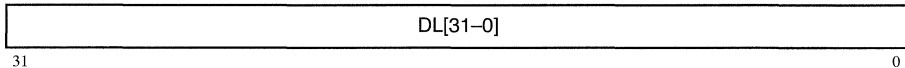


Figure 15-19. Data Low Error Injection Mask (MDP_ERR_INJ_MASK_DL): 4Bytes @ <FF004, F04>

Figure 15-9 shows the bit definitions of the data low error injection mask register

Table 15-9. Data Low Error Injection Mask Bit Field Definitions

Bit	Name	Reset Value	R/W	Description
31-0	DL[31-0]	all 0s	R/W	Error injection mask for memory data path data bus low

15.4.2.3 Parity Error Injection Mask Register

Figure 15-20 shows the bits of the data high error injection mask register and Table 15-10 shows the bit definitions.

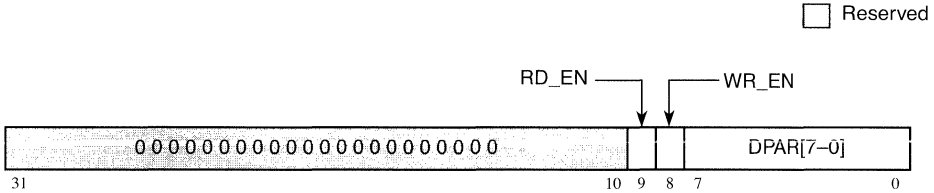


Figure 15-20. Parity Error Injection Mask (MDP_ERR_INJ_MASK_PAR): 4Bytes @ <FF008, F08>

Table 15-10. Parity Error Injection Mask Bit Field Definitions

Bit	Name	Reset Value	R/W	Description
31-8	reserved	all 0s	Read only	reserved
9	RD_EN	0	R/W	Memory data path read enable bit 0 Disables error injection for reads. 1 Enables error injection onto the peripheral logic data bus during reads from local memory.
8	WR_EN	0	R/W	Memory data path write enable bit 0 Disables error injection for writes. 1 Enables error injection onto the memory data bus during writes to local memory.
7-0	DPAR[7-0]	0b0000_0000	R/W	Error injection mask for memory data parity bus.

15.4.3 Memory Data Path Error Capture Monitor Registers

The memory data path error capture monitors are used to latch data that causes ECC or parity errors from either the internal peripheral logic bus or the memory data/parity bus. Separate monitors are provided for the high data, low data, and parity buses. The monitors are read-only. They are loaded by hardware when the error detection registers detects any of the following:

- A memory read parity error / single-bit ECC error trigger exceeded; EDR1[2].
- A multi-bit ECC error; EDR2[3].
- A write parity error; EDR2[2].

When memory data path parity/ECC error data is loaded into the monitors, the capture flag in the MDP_ERR_CAP_MON_PAR is also set. This control bit is sticky and will remain set until manually cleared by the user. The set capture flag prevents subsequent errors from overwriting the data from the first failure. The capture flag is the only bit in the memory data path error capture monitors that is read/write.

15.4.3.1 Data High Error Capture Monitor Register

Figure 15-21 shows the bits of the data high error capture monitor register.

 Reserved

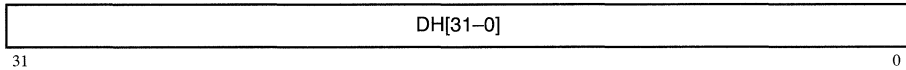


Figure 15-21. Data High Error Capture Monitor (MDP_ERR_CAP_MON_DH): 4Bytes @ <FF00c, F0c>

Table 15-11 shows the bit definitions of the data high error capture monitor register.

Table 15-11. Data High Error Capture Monitor Bit Field Definitions

Bit	Name	Reset Value	R/W	Description
31-0	DH[31-0]	all 0s	Read Only	Capture monitor for memory data path data bus high

15.4.3.2 Data Low Error Capture Monitor Register

Figure 15-22 shows the bits of the data low error capture monitor register.

 Reserved

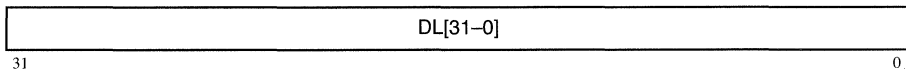


Figure 15-22. Data Low Error Capture Monitor (MDP_ERR_CAP_MON_DL): 4Bytes @ <FF010, F10>


Table 15-12 shows the bit definitions of the data low error capture monitor register.

Table 15-12. Data Low Error Capture Monitor Bit Field Definitions

Bit	Name	Reset Value	R/W	Description
31-0	DL[31-0]	all 0s	Read only	Capture monitor for memory data path data bus low

15.4.3.3 Parity Error Capture Monitor Register

Figure 15-23 shows the bits of the parity error capture monitor register and Table 15-13 shows the bit definitions.

 Reserved

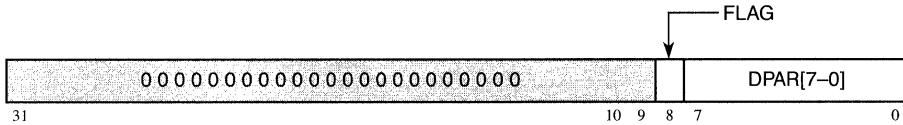


Figure 15-23. Parity Error Capture Monitor (MDP_ERR_CAP_MON_PAR): 4Bytes @ <FF014, F14>

Table 15-13. Parity High Error Capture Monitor Bit Field Definitions

Bit	Name	Reset Value	R/W	Description
31-9	—	all 0s	Read Only	reserved
8	FLAG	0	R/W	Capture flag 0 No data captured 1 Data valid
7-0	DPAR[7-0]	0b0000_0000	Read only	Capture monitor for memory data path data parity bus.

15.5 JTAG/Testing Support

The MPC8240 provides a joint test action group (JTAG) interface to facilitate boundary-scan testing. The JTAG interface is compliant 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, and a test access port (TAP) controller, described in the following sections. A block diagram of the JTAG interface is shown in Figure 15-24.

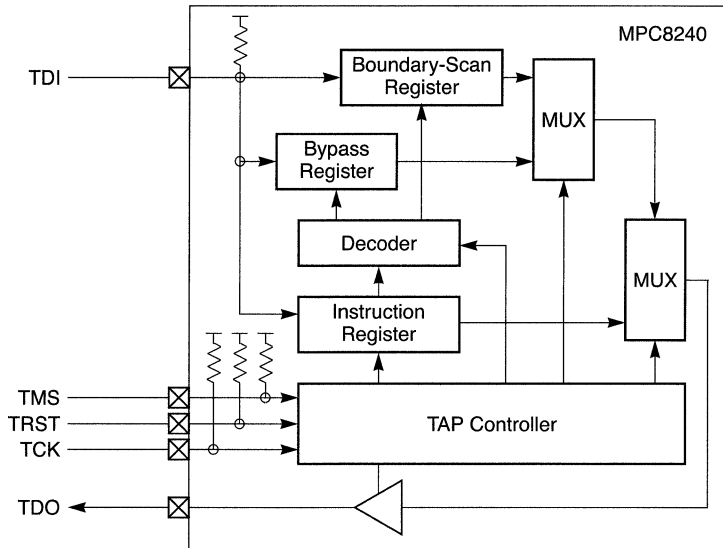


Figure 15-24. JTAG Interface Block Diagram

15.5.1 JTAG Signals

The MPC8240 provides five dedicated JTAG signals—test data input (TDI), test mode select (TMS), test reset ($\overline{\text{TRST}}$), test clock (TCK), and test data output (TDO). The TDI and TDO signals are used to input and output instructions and data to the JTAG scan registers. The boundary-scan operations are controlled by the TAP controller through commands received by means of the TMS signal. 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 assures that the JTAG logic does not interfere with the normal operation of the MPC8240.

Section 3.2.6, “Test and Configuration Signals”, provides more detailed information on the JTAG signals.

15.5.2 JTAG Registers and Scan Chains

The bypass, boundary-scan, and instruction JTAG registers and their associated scan chains are implemented by the MPC8240. These registers are mandatory for compliance with the IEEE 1149.1 specification.

15.5.2.1 Bypass Register

The bypass register is a single-stage register used to bypass the boundary-scan latches of the MPC8240 during board-level boundary-scan operations involving components other than the MPC8240. The use of the bypass register reduces the total scan string size of the boundary-scan test.

15.5.2.2 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 MPC8240. 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 MPC8240'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.

15.5.2.3 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.

15.5.2.4 TAP Controller

The MPC8240 provides a standard JTAG TAP controller that controls instruction and data scan operations. The TMS signal controls the state transitions of the TAP controller.

Appendix A

PowerPC Instruction Set Listings

This appendix lists the MPC8240 microprocessor’s instruction set as well as the additional PowerPC instructions not implemented in the MPC8240. Instructions are sorted by mnemonic, opcode, function, and form. Also included in this appendix is a quick reference table that contains general information, such as the architecture level, privilege level, and form, and indicates if the instruction is 64-bit and optional.



Note that split fields, that represent the concatenation of sequences from left to right, are shown in lowercase. For more information refer to Chapter 8, “Instruction Set,” in *The Programming Environments Manual*.

A.1 Instructions Sorted by Mnemonic

Table A-1 lists the instructions implemented in the PowerPC architecture in alphabetical order by mnemonic.

Table A-1. Complete Instruction List Sorted by Mnemonic

Key:

 Reserved bits  Instruction not implemented in the MPC8240

Name	0	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	
addx	31			D						A								OE									266	Rc
addcx	31			D						A								OE									10	Rc
addex	31			D						A								OE									138	Rc
addi	14			D						A																		SIMM
addic	12			D						A																		SIMM
addic.	13			D						A																		SIMM
addis	15			D						A																		SIMM
addmex	31			D						A						00000		OE									234	Rc
addzex	31			D						A						00000		OE									202	Rc
andx	31			S						A																	28	Rc
andcx	31			S						A																	60	Rc

Instructions Sorted by Mnemonic

Name	0	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
andi.	28	S			A			UIMM																			
andis.	29	S			A			UIMM																			
bx	18	LI																							AA	LK	
bcx	16	BO			BI			BD													AA	LK					
bcctrx	19	BO			BI			0 0 0 0 0	528													LK					
bclrx	19	BO			BI			0 0 0 0 0	16													LK					
cmp	31	crfD	0	L	A			B			0													0			
cmpi	11	crfD	0	L	A			SIMM																			
cmpl	31	crfD	0	L	A			B			32													0			
cmpli	10	crfD	0	L	A			UIMM																			
cntlzdx ⁴	31	S			A			0 0 0 0 0	58													Rc					
cntlzwx	31	S			A			0 0 0 0 0	26													Rc					
crand	19	crbD			crbA			crbB			257													0			
crandc	19	crbD			crbA			crbB			129													0			
creqv	19	crbD			crbA			crbB			289													0			
crnand	19	crbD			crbA			crbB			225													0			
crnor	19	crbD			crbA			crbB			33													0			
cror	19	crbD			crbA			crbB			449													0			
crorc	19	crbD			crbA			crbB			417													0			
crxor	19	crbD			crbA			crbB			193													0			
dcbf	31	0 0 0 0 0			A			B			86													0			
dcbi ¹	31	0 0 0 0 0			A			B			470													0			
dcbst	31	0 0 0 0 0			A			B			54													0			
dcbt	31	0 0 0 0 0			A			B			278													0			
dcbtst	31	0 0 0 0 0			A			B			246													0			
dcbz	31	0 0 0 0 0			A			B			1014													0			
divdx ⁴	31	D			A			B			OE	489													Rc		
divdux ⁴	31	D			A			B			OE	457													Rc		
divwx	31	D			A			B			OE	491													Rc		
divwux	31	D			A			B			OE	459													Rc		
eciwx	31	D			A			B			310													0			
ecowx	31	S			A			B			438													0			
eieio	31	0 0 0 0 0			0 0 0 0 0			0 0 0 0 0			854													0			

Instructions Sorted by Mnemonic

Name	0	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	
eqvx	31		S						A					B														Rc
extsbx	31		S						A					00000														Rc
extshx	31		S						A					00000														Rc
extswx ⁴	31		S						A					00000														Rc
fabsx ⁷	63		D						00000					B														Rc
faddx	63		D						A					B					00000					21			Rc	
faddsx ⁷	59		D						A					B					00000					21			Rc	
fcfidx ^{4,7}	63		D						00000					B														Rc
fcmpo ⁷	63		crfD		00				A					B														0
fcmpu ⁷	63		crfD		00				A					B														0
fcidx ^{4,7}	63		D						00000					B														Rc
fcidzx ^{4,7}	63		D						00000					B														Rc
fcwix ⁷	63		D						00000					B														Rc
fcwizx ⁷	63		D						00000					B														Rc
fdivx ⁷	63		D						A					B					00000					18			Rc	
fdivsx ⁷	59		D						A					B					00000					18			Rc	
fmaddx ⁷	63		D						A					B					C					29			Rc	
fmaddsx ⁷	59		D						A					B					C					29			Rc	
fmr ⁷	63		D						00000					B														Rc
fmsubx ⁷	63		D						A					B					C					28			Rc	
fmsubsx ⁷	59		D						A					B					C					28			Rc	
fmulx ⁷	63		D						A					00000					C					25			Rc	
fmulsx ⁷	59		D						A					00000					C					25			Rc	
fnabsx ⁷	63		D						00000					B														Rc
fnegx ⁷	63		D						00000					B														Rc
fnmaddx ⁷	63		D						A					B					C					31			Rc	
fnmaddsx ⁷	59		D						A					B					C					31			Rc	
fnmsubx ⁷	63		D						A					B					C					30			Rc	
fnmsubsx ⁷	59		D						A					B					C					30			Rc	
fresx ^{5,7}	59		D						00000					B					00000					24			Rc	
frsp ⁷	63		D						00000					B														Rc
frsqrtex ^{5,7}	63		D						00000					B					00000					26			Rc	
fselx ^{5,7}	63		D						A					B					C					23			Rc	

Instructions Sorted by Mnemonic

Name	0	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
fsqrtx ^{5,7}	63	D	0	0	0	0	0	B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	22	Rc			
fsqrtsx ^{5,7}	59	D	0	0	0	0	0	B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	22	Rc			
fsubx ⁷	63	D	A					B	0	0	0	0	0	0	0	0	0	0	0	0	0	20	Rc				
fsubsx ⁷	59	D	A					B	0	0	0	0	0	0	0	0	0	0	0	0	0	20	Rc				
icbi	31	0	0	0	0	0	A	B														982	0				
isync	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	150	0				
lbz	34	D	A																			d					
lbzu	35	D	A																			d					
lbzux	31	D	A					B														119	0				
lbzx	31	D	A					B														87	0				
ld ⁴	58	D	A																			ds		0			
ldarx ⁴	31	D	A					B														84	0				
ldu ⁴	58	D	A																			ds		1			
ldux ⁴	31	D	A					B														53	0				
ldx ⁴	31	D	A					B														21	0				
lfd ⁷	50	D	A																			d					
lfd⁷	51	D	A																			d					
lfd⁷	31	D	A					B														631	0				
lfd⁷	31	D	A					B														599	0				
lfs ⁷	48	D	A																			d					
lfs⁷	49	D	A																			d					
lfs⁷	31	D	A					B														567	0				
lfs⁷	31	D	A					B														535	0				
lha	42	D	A																			d					
lhau	43	D	A																			d					
lh^{aux}	31	D	A					B														375	0				
lh^{ax}	31	D	A					B														343	0				
lh^{brx}	31	D	A					B														790	0				
lh^z	40	D	A																			d					
lh^{zu}	41	D	A																			d					
lh^{zux}	31	D	A					B														311	0				
lh^{zx}	31	D	A					B														279	0				
lmw ³	46	D	A																			d					

Instructions Sorted by Mnemonic

Name	0	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
lswi ³	31			D					A					NB							597						0
lswx ³	31			D					A					B							533						0
lwa ⁴	58			D					A											ds							2
lwarx	31			D					A					B							20						0
lwaux ⁴	31			D					A					B							373						0
lwx ⁴	31			D					A					B							341						0
lwbrx	31			D					A					B							534						0
lwz	32			D					A												d						
lwzu	33			D					A												d						
lwzux	31			D					A					B							55						0
lwzx	31			D					A					B							23						0
mcrf	19			crfD	00			crfS	00					00000							0						0
mcrfs ⁷	63			crfD	00			crfS	00					00000							64						0
mcrxr	31			crfD	00			00000						00000							512						0
mfcrr	31			D				00000						00000							19						0
mffs ⁷	63			D				00000						00000							583						Rc
mfmsr ¹	31			D				00000						00000							83						0
mfspr ²	31			D										spr							339						0
mfsr ¹	31			D	0			SR						00000							595						0
mfsrin ¹	31			D				00000						B							659						0
mftb	31			D										tbr							371						0
mtcrf	31			S	0									CRM							144						0
mtfsb0 ^{x7}	63			crbD				00000						00000							70						Rc
mtfsb1 ^{x7}	63			crbD				00000						00000							38						Rc
mtfsf ⁷	63	0						FM		0				B							711						Rc
mtfsfi ⁷	63			crfD	00			00000						IMM	0						134						Rc
mtmsr ¹	31			S				00000						00000							146						0
mtspr ²	31			S										spr							467						0
mtsr ¹	31			S	0			SR						00000							210						0
mtsrin ¹	31			S				00000						B							242						0
mulhdx ⁴	31			D					A					B	0						73						Rc
mulhdux ⁴	31			D					A					B	0						9						Rc
mulhw ^x	31			D					A					B	0						75						Rc

Instructions Sorted by Mnemonic

Name	0	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	
mulhw_x	31		D			A								B		0					11							Rc
mulld_x ⁴	31		D			A								B		OE					233							Rc
mulli	7		D			A															SIMM							
mullw_x	31		D			A								B		OE					235							Rc
nand_x	31		S			A								B							476							Rc
neg_x	31		D			A							00000			OE					104							Rc
nor_x	31		S			A								B							124							Rc
or_x	31		S			A								B							444							Rc
orc_x	31		S			A								B							412							Rc
ori	24		S			A															UIMM							
oris	25		S			A															UIMM							
rfi ¹	19			00000		00000						00000									50						0	
rlc_x ⁴	30		S			A								B							mb		8					Rc
rlc_r ⁴	30		S			A								B							me		9					Rc
rlc_i ⁴	30		S			A								sh							mb		2		sh			Rc
rlc_l ⁴	30		S			A								sh							mb		0		sh			Rc
rlc_r ⁴	30		S			A								sh							me		1		sh			Rc
rlc_m ⁴	30		S			A								sh							mb		3		sh			Rc
rlwim_x	20		S			A								SH							MB		ME					Rc
rlwinm_x	21		S			A								SH							MB		ME					Rc
rlwnm_x	23		S			A								B							MB		ME					Rc
sc	17			00000		00000															0000000000000000						1	0
slbia ^{1,4,5}	31			00000		00000								00000							498							0
slbie ^{1,4,5}	31			00000		00000								B							434							0
sld_x ⁴	31		S			A								B							27							Rc
slw_x	31		S			A								B							24							Rc
srad_x ⁴	31		S			A								B							794							Rc
srad_i ⁴	31		S			A								sh							413				sh			Rc
sraw_x	31		S			A								B							792							Rc
sraw_i	31		S			A								SH							824							Rc
srd_x ⁴	31		S			A								B							539							Rc
srw_x	31		S			A								B							536							Rc
stb	38		S			A															d							

Instructions Sorted by Mnemonic

Name	0	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	
stbu	39			S					A																			
stbux	31			S					A					B								247						0
stbx	31			S					A					B								215						0
std ⁴	62			S					A																			0
stdcx ⁴	31			S					A					B								214						1
stdu ⁴	62			S					A																			1
stdux ⁴	31			S					A					B									181					0
stdx ⁴	31			S					A					B									149					0
stfd	54			S					A																			
stfdu	55			S					A																			
stfdux	31			S					A					B									759					0
stfdx	31			S					A					B									727					0
stfiwx ⁵	31			S					A					B									983					0
stfs	52			S					A																			
stfsu	53			S					A																			
stfsux	31			S					A					B									695					0
stfsx	31			S					A					B									663					0
sth	44			S					A																			
sthbrx	31			S					A					B									918					0
sthu	45			S					A																			
sthux	31			S					A					B									439					0
sthx	31			S					A					B									407					0
stmw ³	47			S					A																			
stswi ³	31			S					A					NB									725					0
stswx ³	31			S					A					B									661					0
stw	36			S					A																			
stwbrx	31			S					A					B									662					0
stwcx	31			S					A					B									150					1
stwu	37			S					A																			
stwux	31			S					A					B									183					0
stwx	31			S					A					B									151					0
subfx	31			D					A					B									OE	40				Rc
subfcx	31			D					A					B									OE	8				Rc

Instructions Sorted by Mnemonic

Name	0	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
subfex	31	D	A	B	OE	136										Rc											
subfic	08	D	A	SIMM																							
subfmax	31	D	A	00000	OE	232										Rc											
subfzex	31	D	A	00000	OE	200										Rc											
sync	31	00000	00000	00000	598										0												
td ⁴	31	TO	A	B	68										0												
tdi ⁴	02	TO	A	SIMM																							
tlbia ^{1,5}	31	00000	00000	00000	370										0												
tlbie ^{1,5}	31	00000	00000	B	306										0												
tlbid ^{1,6}	31	00000	00000	B	978										0												
tbli ^{1,6}	31	00000	00000	B	1010										0												
tlbsync ^{1,5}	31	00000	00000	00000	566										0												
tw	31	TO	A	B	4										0												
twi	03	TO	A	SIMM																							
xorx	31	S	A	B	316										Rc												
xori	26	S	A	UIMM																							
xoris	27	S	A	UIMM																							

¹ Supervisor-level instruction

² Supervisor- and user-level instruction

³ Load and store string or multiple instruction

⁴ 64-bit instruction

⁵ Optional in the PowerPC architecture

⁶ Implementation-specific instruction

A.2 Instructions Sorted by Opcode

Table A-2 lists the instructions defined in the PowerPC architecture in numeric order by opcode.

Key:



Reserved bits



Instruction not implemented in the MPC8240

Table A-2. Complete Instruction List Sorted by Opcode

Name	0	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		
tdi ⁴	000010	TO			A			SIMM																						
twi	000011	TO			A			SIMM																						
mulli	000111	D			A			SIMM																						
subfic	001000	D			A			SIMM																						
cmpli	001010	crfD	0	L	A			UIMM																						
cmpi	001011	crfD	0	L	A			SIMM																						
addic	001100	D			A			SIMM																						
addic.	001101	D			A			SIMM																						
addi	001110	D			A			SIMM																						
addis	001111	D			A			SIMM																						
bcx	010000	BO			BI			BD																			AA	LK		
sc	010001	00000			00000			0000000000000000																			1	0		
bx	010010	LI																								AA	LK			
mcrf	010011	crfD	00	crfS	00	00000			0000000000																			0		
bclrx	010011	BO			BI			00000			0000010000																			LK
crnor	010011	crbD			crbA			crbB			0000100001																			0
rfi	010011	00000			00000			00000			0000110010																			0
crandc	010011	crbD			crbA			crbB			0010000001																			0
isync	010011	00000			00000			00000			0010010110																			0
crxor	010011	crbD			crbA			crbB			0011000001																			0
crnand	010011	crbD			crbA			crbB			0011100001																			0
crand	010011	crbD			crbA			crbB			0100000001																			0
creqv	010011	crbD			crbA			crbB			0100100001																			0
crorc	010011	crbD			crbA			crbB			0110100001																			0
cror	010011	crbD			crbA			crbB			0111000001																			0
bcctrx	010011	BO			BI			00000			1000010000																			LK

Instructions Sorted by Opcode

Name	0	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
rlwimx	010100	S								A					SH					MB					ME			Rc
rlwinmx	010101	S								A					SH					MB					ME			Rc
rlwnmx	010111	S								A					B					MB					ME			Rc
ori	011000	S								A					UIMM													
oris	011001	S								A					UIMM													
xori	011010	S								A					UIMM													
xoris	011011	S								A					UIMM													
andi	011100	S								A					UIMM													
andis	011101	S								A					UIMM													
rdicx ₄	011110	S								A					sh					mb				000		sh		Rc
rdicrx ₄	011110	S								A					sh					me				001		sh		Rc
rdicx ₄	011110	S								A					sh					mb				010		sh		Rc
rdimix ₄	011110	S								A					sh					mb				011		sh		Rc
rdicx ₄	011110	S								A					B					mb				01000				Rc
rdicrx ₄	011110	S								A					B					me				01001				Rc
cmp	011111	crfD	0	L						A					B					0000000000						0		
tw	011111	TO								A					B					000000100						0		
subfcx	011111	D								A					B	OE				0000001000						Rc		
mulhdux ₄	011111	D								A					B	0				0000001001						Rc		
addcx	011111	D								A					B	OE				0000001010						Rc		
mulhwux	011111	D								A					B	0				0000001011						Rc		
mfcx	011111	D								00000					00000					0000010011						0		
lwarx	011111	D								A					B					0000010100						0		
ldx ₄	011111	D								A					B					0000010101						0		
lwzcx	011111	D								A					B					0000010111						0		
slwxc	011111	S								A					B					0000011000						Rc		

Instructions Sorted by Opcode

Name	0	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
cntlzwx	011111	S								A			00000								0000011010							Rc
sd₄	011111	S								A			B								0000011011							Rc
andx	011111	S								A			B								0000011100							Rc
cmpl	011111	crfD	0	L						A			B								0000100000							0
subfx	011111	D								A			B	OE							0000101000							Rc
ldux⁴	011111	D								A			B								0000110101							0
dcbst	011111	00000								A			B								0000110110							0
lwzux	011111	D								A			B								0000110111							0
cntlzdx₄	011111	S								A			00000								0000111010							Rc
andcx	011111	S								A			B								0000111100							Rc
td⁴	011111	TO								A			B								0001000100							0
mulhdx⁴	011111	D								A			B	0							0001001001							Rc
mulhwx	011111	D								A			B	0							0001001011							Rc
mfmsr	011111	D				00000				00000			00000								0001010011							0
ldarx⁴	011111	D								A			B								0001010100							0
dcbf	011111	00000								A			B								0001010110							0
lbzx	011111	D								A			B								0001010111							0
negx	011111	D								A			00000	OE							0001101000							Rc
lbzux	011111	D								A			B								0001101011							0
norx	011111	S								A			B								0001111100							Rc
subfex	011111	D								A			B	OE							0010001000							Rc
addex	011111	D								A			B	OE							0010001010							Rc
mtrcf	011111	S	0							CRM			0								0010010000							0
mtmsr	011111	S				00000				00000			00000								0010010010							0
stdx⁴	011111	S								A			B								0010010101							0
stwcx.	011111	S								A			B								0010010110							1
stwx	011111	S								A			B								0010010111							0
stdux⁴	011111	S								A			B								0010110101							0
stwux	011111	S								A			B								0010110111							0
subfzex	011111	D								A			00000	OE							0011001000							Rc
addzex	011111	D								A			00000	OE							0011001010							Rc

Instructions Sorted by Opcode

Name	0	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
mtsr	0111111	S	0						SR			00000										0011010010						
stdcx ⁴	0111111	S							A				B									0011010110						1
stbx	0111111	S							A				B									0011010111						0
subfmx	0111111	D							A			00000			OE							0011101000						Rc
muld ⁴	0111111	D							A				B		OE							0011101001						Rc
addmex	0111111	D							A			00000			OE							0011101010						Rc
mullwx	0111111	D							A				B		OE							0011101011						Rc
mtsrin	0111111	S						00000					B									0011110010						0
dcbtst	0111111		00000						A				B									0011110110						0
stbux	0111111	S							A				B									0011110111						0
addx	0111111	D							A				B		OE							0100001010						Rc
dcbt	0111111		00000						A				B									0100010110						0
lhzx	0111111	D							A				B									0100010111						0
eqvx	0111111	S							A				B									0100011100						Rc
tlbie ^{1,5}	0111111		00000				00000						B									0100110010						0
eciwx	0111111	D							A				B									0100110110						0
lhzux	0111111	D							A				B									0100110111						0
xorx	0111111	S							A				B									0100111100						Rc
mfspr ²	0111111	D											spr									0101010011						0
lwax ⁴	0111111	D							A				B									0101010101						0
lhax	0111111	D							A				B									0101010111						0
tlbia ^{1,5}	0111111		00000				00000					00000										0101110010						0
mftb	0111111	D											tbr									0101110011						0
lwaux ⁴	0111111	D							A				B									0101110101						0
lhaux	0111111	D							A				B									0101110111						0
sthx	0111111	S							A				B									0110010111						0
orcx	0111111	S							A				B									0110011100						Rc
sradix ⁴	0111111	S							A				sh									1100111011			sh			Rc
slbie ^{1,4,5}	0111111		00000				00000						B									0110110010						0
ecowx	0111111	S							A				B									0110110110						0
sthux	0111111	S							A				B									0110110111						0
orx	0111111	S							A				B									0110111100						Rc
divdux ⁴	0111111	D							A				B		OE							0111001001						Rc

Instructions Sorted by Opcode

Name	0	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
divwux	0111111				D					A				B				OE			0111001011							Rc
mtspr ²	0111111				S					spr											0111010011							0
dcbi	0111111				00000					A				B							0111010110							0
nandx	0111111				S					A				B							0111011100							Rc
divdx ⁴	0111111				D					A				B				OE			0111101001							Rc
divwx	0111111				D					A				B				OE			0111101011							Rc
sbia ^{1,4,5}	0111111				00000			00000		00000			00000								0111110010							0
mcrxr	0111111				crfD	00		00000		00000			00000								1000000000							0
lswx ³	0111111				D					A				B							1000010101							0
lwbx	0111111				D					A				B							1000010110							0
lfsx ⁷	0111111				D					A				B							1000010111							0
srwx	0111111				S					A				B							1000011000							Rc
srdx ⁴	0111111				S					A				B							1000011011							Rc
tlbsync ^{1,5}	0111111				00000			00000		00000			00000								1000110110							0
lfsux ⁷	0111111				D					A				B							1000110111							0
mfsr	0111111				D	0		SR		00000											1001010011							0
lswi ³	0111111				D					A				NB							1001010101							0
sync	0111111				00000			00000		00000			00000								1001010110							0
lfdx ⁷	0111111				D					A				B							1001010111							0
lfdx ⁷	0111111				D					A				B							1001110111							0
mfsrin ¹	0111111				D			00000						B							1010010011							0
stswx ³	0111111				S					A				B							1010010101							0
stwbrx	0111111				S					A				B							1010010110							0
stfsx	0111111				S					A				B							1010010111							0
stfsux	0111111				S					A				B							1010110111							0
stswi ³	0111111				S					A				NB							1011010101							0
stfdx ⁷	0111111				S					A				B							1011010111							0
stfdx ⁷	0111111				S					A				B							1011110111							0
lhbrx	0111111				D					A				B							1100010110							0
srawx	0111111				S					A				B							1100011000							Rc
sradx ⁴	0111111				S					A				B							1100011010							Rc
srawix	0111111				S					A				SH							1100111000							Rc
eieio	0111111				00000			00000		00000			00000								1101010110							0

Instructions Sorted by Opcode

Name	0	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
sthbrx	0111111	S								A																		0
extshx	0111111	S								A			00000															Rc
extsbx	0111111	S								A			00000															Rc
tblld ^{1,6}	0111111		00000										00000															0
icbi	0111111		00000							A																		0
stfiwx ⁵	0111111	S								A																		0
extsw ⁴	0111111	S								A			00000															Rc
tbli ^{1,6}	0111111		00000										00000															0
dcbz	0111111		00000							A																		0
lwz	100000									D																		
lwzu	100001									D																		
lbz	100010									D																		
lbzu	100011									D																		
stw	100100									S																		
stwu	100101									S																		
stb	100110									S																		
stbu	100111									S																		
lhz	101000									D																		
lhzu	101001									D																		
lha	101010									D																		
lhau	101011									D																		
sth	101100									S																		
sthu	101101									S																		
lmw ³	101110									D																		
stmw ³	101111									S																		
lfs ⁷	110000									D																		
lfsu ⁷	110001									D																		
lfd ⁷	110010									D																		
lfd_u ⁷	110011									D																		
stfs ⁷	110100									S																		
stfs_u ⁷	110101									S																		
stfd ⁷	110110									S																		
stfd_u ⁷	110111									S																		

Instructions Sorted by Opcode

Name 0 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

ld ⁴	1 1 1 0 1 0	D	A	ds				0 0
ldu ⁴	1 1 1 0 1 0	D	A	ds				0 1
lwa ⁴	1 1 1 0 1 0	D	A	ds				1 0
fdivsx	1 1 1 0 1 1	D	A	B	0 0 0 0 0	1 0 0 1 0	Rc	
fsubsx	1 1 1 0 1 1	D	A	B	0 0 0 0 0	1 0 1 0 0	Rc	
faddsx	1 1 1 0 1 1	D	A	B	0 0 0 0 0	1 0 1 0 1	Rc	
fsqrtx ^{5,7}	1 1 1 0 1 1	D	0 0 0 0 0	B	0 0 0 0 0	1 0 1 1 0	Rc	
fresx ^{5,7}	1 1 1 0 1 1	D	0 0 0 0 0	B	0 0 0 0 0	1 1 0 0 0	Rc	
fmulsx	1 1 1 0 1 1	D	A	0 0 0 0 0	C	1 1 0 0 1	Rc	
fmsubsx ⁷	1 1 1 0 1 1	D	A	B	C	1 1 1 0 0	Rc	
fmaddsx ⁷	1 1 1 0 1 1	D	A	B	C	1 1 1 0 1	Rc	
fnmsubsx	1 1 1 0 1 1	D	A	B	C	1 1 1 1 0	Rc	
fnmaddsx	1 1 1 0 1 1	D	A	B	C	1 1 1 1 1	Rc	
std ⁴	1 1 1 1 1 0	S	A	ds				0 0
stdu ⁴	1 1 1 1 1 0	S	A	ds				0 1
fcmpu ⁷	1 1 1 1 1 1	crfD	0 0	A	B	0 0 0 0 0 0 0 0 0	0	
frspx	1 1 1 1 1 1	D	0 0 0 0 0	B	0 0 0 0 0 0 1 1 0 0	Rc		
fctiw	1 1 1 1 1 1	D	0 0 0 0 0	B	0 0 0 0 0 0 1 1 1 0			
fctiwz	1 1 1 1 1 1	D	0 0 0 0 0	B	0 0 0 0 0 0 1 1 1 1	Rc		
fdiv	1 1 1 1 1 1	D	A	B	0 0 0 0 0	1 0 0 1 0	Rc	
fsub	1 1 1 1 1 1	D	A	B	0 0 0 0 0	1 0 1 0 0	Rc	
fadd	1 1 1 1 1 1	D	A	B	0 0 0 0 0	1 0 1 0 1	Rc	
fsqrt ^{5,7}	1 1 1 1 1 1	D	0 0 0 0 0	B	0 0 0 0 0	1 0 1 1 0	Rc	
fsel ^{5,7}	1 1 1 1 1 1	D	A	B	C	1 0 1 1 1	Rc	
fmul	1 1 1 1 1 1	D	A	0 0 0 0 0	C	1 1 0 0 1	Rc	
frsqrt ^{5,7}	1 1 1 1 1 1	D	0 0 0 0 0	B	0 0 0 0 0	1 1 0 1 0	Rc	
fmsub	1 1 1 1 1 1	D	A	B	C	1 1 1 0 0	Rc	
fmadd	1 1 1 1 1 1	D	A	B	C	1 1 1 0 1	Rc	
fnmsub	1 1 1 1 1 1	D	A	B	C	1 1 1 1 0	Rc	
fnmadd ⁷	1 1 1 1 1 1	D	A	B	C	1 1 1 1 1	Rc	
fcmpo ⁷	1 1 1 1 1 1	crfD	0 0	A	B	0 0 0 0 1 0 0 0 0 0	0	
mtfsb1 _x	1 1 1 1 1 1	crbD	0 0 0 0 0	0 0 0 0 0	0 0 0 0 1 0 0 1 1 0	Rc		
fneg _x	1 1 1 1 1 1	D	0 0 0 0 0	B	0 0 0 0 1 0 1 0 0 0	Rc		

Instructions Sorted by Opcode

Name	0	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		
mcrfs ⁷	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	
mtfsb0x	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	0	0	1	1
fmr	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
mtfsfix	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
fnabsx	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
fabsx	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
mffsx	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
mtfsfx	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
fctidx ^{4,7}	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
fctidzx ^{4,7}	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
fcfidx ^{4,7}	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	

- ¹ Supervisor-level instruction
- ² Supervisor- and user-level instruction
- ³ Load and store string or multiple instruction
- ⁴ 64-bit instruction
- ⁵ Optional in the PowerPC architecture
- ⁶ MPC8240-implementation specific instruction

A.3 Instructions Grouped by Functional Categories

Table A-3 through Table A-30 list the PowerPC instructions grouped by function.

Key:



Reserved bits



Instruction not implemented in the MPC8240

Table A-3. Integer Arithmetic Instructions

Name	0	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
addx	31				D					A								B	OE				266					Rc
addcx	31				D					A								B	OE				10					Rc
addex	31				D					A								B	OE				138					Rc
addi	14				D					A													SIMM					
addic	12				D					A													SIMM					
addic.	13				D					A													SIMM					
addis	15				D					A													SIMM					
addmex	31				D					A								00000	OE				234					Rc
addzex	31				D					A								00000	OE				202					Rc
divdx ⁴	31				D					A								B	OE				489					Rc
divdux ⁴	31				D					A								B	OE				457					Rc
divwx	31				D					A								B	OE				491					Rc
divwux	31				D					A								B	OE				459					Rc
mulhdx ⁴	31				D					A								B	0				73					Rc
mulhdwx ⁴	31				D					A								B	0				9					Rc
mulhwx	31				D					A								B	0				75					Rc
mulhwux	31				D					A								B	0				11					Rc
mulld ⁴	31				D					A								B	OE				233					Rc
mulldi	07				D					A													SIMM					
mullwx	31				D					A								B	OE				235					Rc
negx	31				D					A								00000	OE				104					Rc
subfx	31				D					A								B	OE				40					Rc
subfcx	31				D					A								B	OE				8					Rc
subficx	08				D					A													SIMM					
subfex	31				D					A								B	OE				136					Rc
subfmex	31				D					A								00000	OE				232					Rc
subfzex	31				D					A								00000	OE				200					Rc

Instructions Grouped by Functional Categories

Table A-4. Integer Compare Instructions

Name	0	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
cmp	31	crfD	0	L		A		B	0000000000						0													
cmpi	11	crfD	0	L		A	SIMM																					
cmpl	31	crfD	0	L		A		B	32						0													
cmpli	10	crfD	0	L		A	UIMM																					

Table A-5. Integer Logical Instructions

Name	0	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
andx	31		S			A		B	28						Rc													
andcx	31		S			A		B	60						Rc													
andi.	28		S			A	UIMM																					
andis.	29		S			A	UIMM																					
cntlzdx ⁴	31		S			A	00000	58						Rc														
cntlzwx	31		S			A	00000	26						Rc														
eqvx	31		S			A		B	284						Rc													
extsbx	31		S			A	00000	954						Rc														
extshx	31		S			A	00000	922						Rc														
extswx ⁴	31		S			A	00000	986						Rc														
nandx	31		S			A		B	476						Rc													
norx	31		S			A		B	124						Rc													
orx	31		S			A		B	444						Rc													
orcx	31		S			A		B	412						Rc													
ori	24		S			A	UIMM																					
oris	25		S			A	UIMM																					
xorx	31		S			A		B	316						Rc													
xori	26		S			A	UIMM																					
xoris	27		S			A	UIMM																					

Table A-6. Integer Rotate Instructions

Name	0	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
rldclx ⁴	30		S			A		B	mb	8						Rc												
rldcrx ⁴	30		S			A		B	me	9						Rc												
rldicx ⁴	30		S			A		sh	mb	2						sh	Rc											
rldicl ⁴	30		S			A		sh	mb	0						sh	Rc											

Table A-6. Integer Rotate Instructions (Continued)

rlcicrx ⁴	30	S	A	sh	me	1	sh	Rc
rldimix ⁴	30	S	A	sh	mb	3	sh	Rc
rlwimix	22	S	A	SH	MB	ME		Rc
rlwinmx	20	S	A	SH	MB	ME		Rc
rlwnmx	21	S	A	SH	MB	ME		Rc

Table A-7. Integer Shift Instructions

Name	0	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	
sldx ⁴	31	S	A	B															27										Rc
slwx	31	S	A	B															24										Rc
sradx ⁴	31	S	A	B															794										Rc
sradix ⁴	31	S	A	sh															413						sh				Rc
srawx	31	S	A	B															792										Rc
srawix	31	S	A	SH															824										Rc
srdx ⁴	31	S	A	B															539										Rc
srwx	31	S	A	B															536										Rc

Table A-8. Floating-Point Arithmetic Instructions⁷

Name	0	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	
faddx	63	D	A	B															00000					21					Rc
faddsx	59	D	A	B															00000					21					Rc
fdivx	63	D	A	B															00000					18					Rc
fdivsx	59	D	A	B															00000					18					Rc
fmulx	63	D	A	00000															C					25					Rc
fmulsx	59	D	A	00000															C					25					Rc
fresx ⁵	59	D	00000																00000					24					Rc
frsqrtox ⁵	63	D	00000																00000					26					Rc
fsubx	63	D	A	B															00000					20					Rc
fsubsx	59	D	A	B															00000					20					Rc
fselx ⁵	63	D	A	B															C					23					Rc
fsqrtx ⁵	63	D	00000																00000					22					Rc
fsqrtxs ⁵	59	D	00000																00000					22					Rc

Instructions Grouped by Functional Categories

Table A-9. Floating-Point Multiply-Add Instructions⁷

Name	0	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
fmaddx	63				D					A					B						C					29	Rc	
fmaddsx	59				D					A					B						C					29	Rc	
fmsubx	63				D					A					B						C					28	Rc	
fmsubsx	59				D					A					B						C					28	Rc	
fnmaddx	63				D					A					B						C					31	Rc	
fnmaddsx	59				D					A					B						C					31	Rc	
fnmsubx	63				D					A					B						C					30	Rc	
fnmsubsx	59				D					A					B						C					30	Rc	

Table A-10. Floating-Point Rounding and Conversion Instructions⁷

Name	0	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		
fctid⁴x	63				D				0	0	0	0			B													846	Rc	
fctid⁴	63				D				0	0	0	0			B													814	Rc	
fctidz⁴	63				D				0	0	0	0			B													815	Rc	
fctiw^x	63				D				0	0	0	0			B													14	Rc	
fctiwz^x	63				D				0	0	0	0			B														15	Rc
frsp^x	63				D				0	0	0	0			B														12	Rc

Table A-11. Floating-Point Compare Instructions⁷

Name	0	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		
fcmpo	63				crfD		0	0		A					B													32	0	
fcmpu	63				crfD		0	0		A					B														0	0

Table A-12. Floating-Point Status and Control Register Instructions⁷

Name	0	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		
mcrfs	63				crfD		0	0		crfS		0	0		0	0	0	0	0									64	0	
mffsx	63				D				0	0	0	0			0	0	0	0	0										583	Rc
mtfsb0^x	63				crbD				0	0	0	0			0	0	0	0	0										70	Rc
mtfsb1^x	63				crbD				0	0	0	0			0	0	0	0	0										38	Rc
mtfsfx	31		0							FM		0			B														711	Rc
mtfsfix	63				crfD		0	0		0	0	0	0		IMM		0												134	Rc

Table A-13. Integer Load Instructions

Name	0	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	
lbz	34			D						A																			
lbzu	35			D						A																			
lbzux	31			D						A					B								119					0	
lbzx	31			D						A					B								87					0	
ld ⁴	58			D						A																			0
ldu ⁴	58			D						A																			1
ldux ⁴	31			D						A					B														0
ldx ⁴	31			D						A					B														0
lha	42			D						A																			
lhau	43			D						A																			
lhaux	31			D						A					B														0
lhax	31			D						A					B														0
lhz	40			D						A																			
lhzu	41			D						A																			
lhzux	31			D						A					B														0
lhzx	31			D						A					B														0
lwa ⁴	58			D						A																			2
lwaux ⁴	31			D						A					B														0
lwx ⁴	31			D						A					B														0
lwz	32			D						A																			
lwzu	33			D						A																			
lwzux	31			D						A					B														0
lwzx	31			D						A					B														0

Table A-14. Integer Store Instructions

Name	0	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	
stb	38			S						A																			
stbu	39			S						A																			
stbux	31			S						A					B														0
stbx	31			S						A					B														0
std ⁴	62			S						A																			0
stdu ⁴	62			S						A																			1
stdux ⁴	31			S						A					B														0

Table A-14. Integer Store Instructions (Continued)

stdx ⁴	31	S	A	B	149	0
sth	44	S	A	d		
sth	45	S	A	d		
sthux	31	S	A	B	439	0
sthx	31	S	A	B	407	0
stw	36	S	A	d		
stwu	37	S	A	d		
stwux	31	S	A	B	183	0
stwx	31	S	A	B	151	0

Table A-15. Integer Load and Store with Byte-Reverse Instructions

Name	0	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
lhbrx	31	D	A	B	790			0																				
lwbrx	31	D	A	B	534			0																				
sthbrx	31	S	A	B	918			0																				
stwbrx	31	S	A	B	662			0																				

Table A-16. Integer Load and Store Multiple Instructions

Name	0	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
lmw ³	46	D	A	d																								
stmw ³	47	S	A	d																								

Table A-17. Integer Load and Store String Instructions

Name	0	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
lswi ³	31	D	A	NB	597			0																				
lswx ³	31	D	A	B	533			0																				
stswi ³	31	S	A	NB	725			0																				
stswx ³	31	S	A	B	661			0																				

Table A-18. Memory Synchronization Instructions

Name	0	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
eieio	31	00000	00000	00000	854			0																				
isync	19	00000	00000	00000	150			0																				
ldarx ⁴	31	D	A	B	84			0																				
lwarx	31	D	A	B	20			0																				
stdcx ⁴	31	S	A	B	214			1																				

Table A-18. Memory Synchronization Instructions (Continued)

stwcx.	31	S	A	B	150	1
sync	31	00000	00000	00000	598	0

Table A-19. Floating-Point Load Instructions⁷

Name 0 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

lfd	50	D	A	d		
lfdu	51	D	A	d		
lfdx	31	D	A	B	631	0
lfdx	31	D	A	B	599	0
lfs	48	D	A	d		
lfsu	49	D	A	d		
lfsux	31	D	A	B	567	0
lfsx	31	D	A	B	535	0

Table A-20. Floating-Point Store Instructions⁷

Name 0 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

stfd	54	S	A	d		
stfdu	55	S	A	d		
stfdx	31	S	A	B	759	0
stfdx	31	S	A	B	727	0
stfiwx ⁵	31	S	A	B	983	0
stfs	52	S	A	d		
stfsu	53	S	A	d		
stfsux	31	S	A	B	695	0
stfsx	31	S	A	B	663	0

Table A-21. Floating-Point Move Instructions⁷

Name 0 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

fabsx	63	D	00000	B	264	Rc
fmrx	63	D	00000	B	72	Rc
fnabsx	63	D	00000	B	136	Rc
fnegx	63	D	00000	B	40	Rc

Table A-22. Branch Instructions

Name	0	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
bx	18	LI														AA	LK											
bcx	16	BO			BI			BD											AA	LK								
bcctrx	19	BO			BI			00000				528				LK												
bclrx	19	BO			BI			00000				16				LK												

Table A-23. Condition Register Logical Instructions

Name	0	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
crand	19	crbD			crbA			crbB			257											0						
crandc	19	crbD			crbA			crbB			129											0						
creqv	19	crbD			crbA			crbB			289											0						
crnand	19	crbD			crbA			crbB			225											0						
crnor	19	crbD			crbA			crbB			33											0						
cror	19	crbD			crbA			crbB			449											0						
crorc	19	crbD			crbA			crbB			417											0						
crxor	19	crbD			crbA			crbB			193											0						
mcrf	19	crfD		00		crfS		00		00000				0000000000											0			

Table A-24. System Linkage Instructions

Name	0	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
rfi ¹	19	00000			00000			00000			50											0						
sc	17	00000			00000			0000000000000000											1	0								

Table A-25. Trap Instructions

Name	0	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
td ⁴	31	TO			A			B			68											0						
tdi ⁴	03	TO			A			SIMM																				
tw	31	TO			A			B			4											0						
twi	03	TO			A			SIMM																				

Table A-26. Processor Control Instructions

Name	0	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
mcrxr	31	crfS		00		00000			00000			512											0					
mfcrr	31	D			00000			00000			19											0						
mfmsr ¹	31	D			00000			00000			83											0						
mfmspr ²	31	D			spr											339				0								

Table A-26. Processor Control Instructions (Continued)

mftb	31	D	tpr				371	0	
mtrcf	31	S	0	CRM			0	144	0
mtmsr ¹	31	S	00000		00000		146	0	
mtspr ²	31	D	spr				467	0	

Table A-27. Cache Management Instructions

Name	0	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
dcbf	31	00000				A				B				86				0										
dcbi ¹	31	00000				A				B				470				0										
dcbst	31	00000				A				B				54				0										
dcbt	31	00000				A				B				278				0										
dcbtst	31	00000				A				B				246				0										
dcbz	31	00000				A				B				1014				0										
icbi	31	00000				A				B				982				0										

Table A-28. Segment Register Manipulation Instructions

Name	0	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
mfsr ¹	31	D	0	SR				00000				595				0												
mfsrin ¹	31	D	00000				B				659				0													
mtsr ¹	31	S	0	SR				00000				210				0												
mtsrin ¹	31	S	00000				B				242				0													

Table A-29. Lookaside Buffer Management Instructions

Name	0	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
slbia ^{1,4,5}	31	00000				00000				00000				498				0										
slbie ^{1,4,5}	31	00000				00000				B				434				0										
tlbia ^{1,5}	31	00000				00000				00000				370				0										
tlbie ^{1,5}	31	00000				00000				B				306				0										
tlbid ^{1,6}	31	00000				00000				B				978				0										
tlbli ^{1,6}	31	00000				00000				B				1010				0										
tlbsync ^{1,5}	31	00000				00000				00000				566				0										

Table A-30. External Control Instructions

Name 0 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

eciwx	31	D	A	B	310	0
ecowx	31	S	A	B	438	0

- ¹ Supervisor-level instruction
- ² Supervisor- and user-level instruction
- ³ Load and store string or multiple instruction
- ⁴ 64-bit instruction
- ⁵ Optional in the PowerPC architecture
- ⁶ MPC8240-implementation specific instruction

A.4 Instructions Sorted by Form

Table A-31 through Table A-45 list the PowerPC instructions grouped by form.

Key:

Reserved bits

Instruction not implemented in the MPC8240

Table A-31. I-Form

OPCD	LI	AA	LK
------	----	----	----

Specific Instruction

Name 0 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

bx	18	LI	AA	LK
-----------	----	----	----	----

Table A-32. B-Form

OPCD	BO	BI	BD	AA	LK
------	----	----	----	----	----

Specific Instruction

Name 0 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

bcx	16	BO	BI	BD	AA	LK
------------	----	----	----	----	----	----

Table A-33. SC-Form

OPCD	00000	00000	00000000000000000000	1	0
------	-------	-------	----------------------	---	---

Specific Instruction

Name 0 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

sc	17	00000	00000	00000000000000000000	1	0
-----------	----	-------	-------	----------------------	---	---

Table A-34. D-Form

OPCD	D	A	d	
OPCD	D	A	SIMM	
OPCD	S	A	d	
OPCD	S	A	UIMM	
OPCD	crfD	0 L	A	SIMM
OPCD	crfD	0 L	A	UIMM
OPCD	TO	A	SIMM	

Specific Instructions

Name 0 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

addi	14	D	A	SIMM
addic	12	D	A	SIMM

Instructions Sorted by Form

addc.	13	D	A	SIMM	
addis	15	D	A	SIMM	
andi.	28	S	A	UIMM	
andis.	29	S	A	UIMM	
cmpi	11	crfD	0 L	A	SIMM
cmpli	10	crfD	0 L	A	UIMM
lbz	34	D	A	d	
lbzu	35	D	A	d	
lfd ⁷	50	D	A	d	
lfdu ⁷	51	D	A	d	
lfs ⁷	48	D	A	d	
lfsu ⁷	49	D	A	d	
lha	42	D	A	d	
lhau	43	D	A	d	
lhz	40	D	A	d	
lhzu	41	D	A	d	
lmw ³	46	D	A	d	
lwz	32	D	A	d	
lwzu	33	D	A	d	
mulli	7	D	A	SIMM	
ori	24	S	A	UIMM	
oris	25	S	A	UIMM	
stb	38	S	A	d	
stbu	39	S	A	d	
stfd ⁷	54	S	A	d	
stfdu ⁷	55	S	A	d	
stfs ⁷	52	S	A	d	
stfsu ⁷	53	S	A	d	
sth	44	S	A	d	
sthu	45	S	A	d	
stmw ³	47	S	A	d	
stw	36	S	A	d	
stwu	37	S	A	d	
subfic	08	D	A	SIMM	
tdi ⁴	02	TO	A	SIMM	

twi	03	TO	A	SIMM
xori	26	S	A	UIMM
xoris	27	S	A	UIMM

Table A-35. DS-Form

OPCD	D	A	ds	XO
OPCD	S	A	ds	XO

Specific Instructions

Name 0 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

ld ⁴	58	D	A	ds	0
ldu ⁴	58	D	A	ds	1
lwa ⁴	58	D	A	ds	2
std ⁴	62	S	A	ds	0
stdu ⁴	62	S	A	ds	1

Table A-36. X-Form

OPCD	D	A	B	XO	0		
OPCD	D	A	NB	XO	0		
OPCD	D	00000	B	XO	0		
OPCD	D	00000	00000	XO	0		
OPCD	D	0 SR	00000	XO	0		
OPCD	S	A	B	XO	Rc		
OPCD	S	A	B	XO	1		
OPCD	S	A	B	XO	0		
OPCD	S	A	NB	XO	0		
OPCD	S	A	00000	XO	Rc		
OPCD	S	00000	B	XO	0		
OPCD	S	00000	00000	XO	0		
OPCD	S	0 SR	00000	XO	0		
OPCD	S	A	SH	XO	Rc		
OPCD	crfD	0 L	A	B	XO	0	
OPCD	crfD	00	A	B	XO	0	
OPCD	crfD	00	crfS	00	00000	XO	0
OPCD	crfD	00	00000	00000	XO	0	
OPCD	crfD	00	00000	IMM	0	Rc	
OPCD	TO	A	B	XO	0		

Instructions Sorted by Form

OPCD	D	00000	B	XO	Rc
OPCD	D	00000	00000	XO	Rc
OPCD	crbD	00000	00000	XO	Rc
OPCD	00000	A	B	XO	0
OPCD	00000	00000	B	XO	0
OPCD	00000	00000	00000	XO	0

Specific Instructions

andx	31	S	A	B	28	Rc	
andcx	31	S	A	B	60	Rc	
cmp	31	crfD	0 L	A	B	0	0
cmpl	31	crfD	0 L	A	B	32	0
cntlidx ⁴	31	S	A	00000	58	Rc	
cntlzx	31	S	A	00000	26	Rc	
dcbf	31	00000	A	B	86	0	
dcbi ¹	31	00000	A	B	470	0	
dcbst	31	00000	A	B	54	0	
dcbt	31	00000	A	B	278	0	
dcbtst	31	00000	A	B	246	0	
dcbz	31	00000	A	B	1014	0	
eciw	31	D	A	B	310	0	
ecow	31	S	A	B	438	0	
eieio	31	00000	00000	00000	854	0	
eqvx	31	S	A	B	284	Rc	
extsbx	31	S	A	00000	954	Rc	
extshx	31	S	A	00000	922	Rc	
extswx ⁴	31	S	A	00000	986	Rc	
fabsx	63	D	00000	B	264	Rc	
fctidx ^{4,7}	63	D	00000	B	846	Rc	
fcmpo ⁷	63	crfD	00	A	B	32	0
fcmpu ⁷	63	crfD	00	A	B	0	0
fctid ^{4,7}	63	D	00000	B	814	Rc	
fctidz ^{4,7}	63	D	00000	B	815	Rc	
fctiw	63	D	00000	B	14	Rc	
fctiwZ ⁷	63	D	00000	B	15	Rc	

Instructions Sorted by Form

fmr _x	63	D	00000	B	72	Rc		
fnabs _x	63	D	00000	B	136	Rc		
fneg _x	63	D	00000	B	40	Rc		
frsp _x	63	D	00000	B	12	Rc		
icbi	31	00000	A	B	982	0		
lbz _{ux}	31	D	A	B	119	0		
lbz _x	31	D	A	B	87	0		
ldar _x ⁴	31	D	A	B	84	0		
ldu _x ⁴	31	D	A	B	53	0		
ldx ⁴	31	D	A	B	21	0		
lfd _{ux} ⁷	31	D	A	B	631	0		
lfd _x ⁷	31	D	A	B	599	0		
lfs _{ux} ⁷	31	D	A	B	567	0		
lfs _x ⁷	31	D	A	B	535	0		
lhau _x	31	D	A	B	375	0		
lhax	31	D	A	B	343	0		
lhbr _x	31	D	A	B	790	0		
lhz _{ux}	31	D	A	B	311	0		
lhz _x	31	D	A	B	279	0		
lswi ³	31	D	A	NB	597	0		
lsw _x ³	31	D	A	B	533	0		
lwar _x	31	D	A	B	20	0		
lwa _{ux} ⁴	31	D	A	B	373	0		
lwa _x ⁴	31	D	A	B	341	0		
lwbr _x	31	D	A	B	534	0		
lwz _{ux}	31	D	A	B	55	0		
lwz _x	31	D	A	B	23	0		
mcrfs	63	crfD	00	crfS	00	00000	64	0
mcrxr	31	crfD	00	00000	00000	00000	512	0
mfc _r	31	D	00000	00000	00000	00000	19	0
mffs _x	63	D	00000	00000	00000	00000	583	Rc
mfms _r ¹	31	D	00000	00000	00000	00000	83	0
mfs _r ¹	31	D	0	SR	00000	00000	595	0
mfsrin ¹	31	D	00000	B	00000	00000	659	0
mtfsb0 _x ⁷	63	crbD	00000	00000	00000	00000	70	Rc

Instructions Sorted by Form

mtfsb1x ⁷	63	crfD		00000	00000	38	Rc
mtfsfix	63	crbD	00	00000	IMM 0	134	Rc
mtmsr ¹	31	S		00000	00000	146	0
mtr ¹	31	S	0	SR	00000	210	0
mtsrin ¹	31	S		00000	B	242	0
nandx	31	S		A	B	476	Rc
norx	31	S		A	B	124	Rc
orx	31	S		A	B	444	Rc
orcx	31	S		A	B	412	Rc
slbia ^{1,4,5}	31	00000		00000	00000	498	0
slbie ^{1,4,5}	31	00000		00000	B	434	0
sidx ⁴	31	S		A	B	27	Rc
slwx	31	S		A	B	24	Rc
sradx ⁴	31	S		A	B	794	Rc
srawx	31	S		A	B	792	Rc
srawix	31	S		A	SH	824	Rc
srdx ⁴	31	S		A	B	539	Rc
srwx	31	S		A	B	536	Rc
stbux	31	S		A	B	247	0
stbx	31	S		A	B	215	0
stdcx ⁴	31	S		A	B	214	1
stdux ⁴	31	S		A	B	181	0
stdx ⁴	31	S		A	B	149	0
stfdx ⁷	31	S		A	B	759	0
stfdx ⁷	31	S		A	B	727	0
stfiwx ^{5,7}	31	S		A	B	983	0
stfsux ⁷	31	S		A	B	695	0
stfsx ⁷	31	S		A	B	663	0
sthbrx	31	S		A	B	918	0
sthux	31	S		A	B	439	0
sthx	31	S		A	B	407	0
stswi ³	31	S		A	NB	725	0
stswx ³	31	S		A	B	661	0
stwbrx	31	S		A	B	662	0
stwcx	31	S		A	B	150	1

stwux	31	S	A	B	183	0
stwx	31	S	A	B	151	0
sync	31	0 0 0 0	0 0 0 0	0 0 0 0	598	0
td ⁴	31	TO	A	B	68	0
tibia ^{1,5}	31	0 0 0 0	0 0 0 0	0 0 0 0	370	0
tlbie ^{1,5}	31	0 0 0 0	0 0 0 0	B	306	0
tlbid ^{1,6}	31	0 0 0 0	0 0 0 0	B	978	0
tlbli ^{1,6}	31	0 0 0 0	0 0 0 0	B	1010	0
tlbsync ^{1,5}	31	0 0 0 0	0 0 0 0	0 0 0 0	566	0
tw	31	TO	A	B	4	0
xorx	31	S	A	B	316	Rc

Table A-37. XL-Form

OPCD	BO	BI	0 0 0 0	XO	LK		
OPCD	crbD	crbA	crbB	XO	0		
OPCD	crfD	0 0	crfS	0 0	0 0 0 0	XO	0
OPCD	0 0 0 0	0 0 0 0	0 0 0 0	XO	0		

Specific Instructions

Name	0	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
bcctrx	19	BO	BI	0 0 0 0	528	LK																						
bclrx	19	BO	BI	0 0 0 0	16	LK																						
crand	19	crbD	crbA	crbB	257	0																						
crandc	19	crbD	crbA	crbB	129	0																						
creqv	19	crbD	crbA	crbB	289	0																						
crnand	19	crbD	crbA	crbB	225	0																						
crnor	19	crbD	crbA	crbB	33	0																						
cror	19	crbD	crbA	crbB	449	0																						
crorc	19	crbD	crbA	crbB	417	0																						
crxor	19	crbD	crbA	crbB	193	0																						
isync	19	0 0 0 0	0 0 0 0	0 0 0 0	150	0																						
mcrf	19	crfD	0 0	crfS	0 0	0 0 0 0	0	0																				
rfi ¹	19	0 0 0 0	0 0 0 0	0 0 0 0	50	0																						

Table A-38. XFX-Form

OPCD	D	spr										XO	0		
OPCD	D	0	CRM										0	XO	0
OPCD	S	spr										XO	0		
OPCD	D	tbr										XO	0		

Specific Instructions

Name 0 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

mfspr ²	31	D	spr										339	0		
mftb	31	D	tbr										371	0		
mtrcf	31	S	0	CRM										0	144	0
mtspr ²	31	D	spr										467	0		

Table A-39. XFL-Form

OPCD	0	FM										0	B	XO	Rc
------	---	----	--	--	--	--	--	--	--	--	--	---	---	----	----

Specific Instructions

Name 0 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

mtfsx ⁷	63	0	FM										0	B	711	Rc
---------------------------	----	---	----	--	--	--	--	--	--	--	--	--	---	---	-----	----

Table A-40. XS-Form

OPCD	S	A										sh	XO	sh	Rc
------	---	---	--	--	--	--	--	--	--	--	--	----	----	----	----

Specific Instructions

Name 0 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

sradix ⁴	31	S	A										sh	413	sh	Rc
----------------------------	----	---	---	--	--	--	--	--	--	--	--	--	----	-----	----	----

Table A-41. XO-Form

OPCD	D	A										B	OE	XO	Rc
OPCD	D	A										B	0	XO	Rc
OPCD	D	A										0 0 0 0 0	OE	XO	Rc

Specific Instructions

Name 0 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

addx	31	D	A										B	OE	266	Rc
addcx	31	D	A										B	OE	10	Rc
addex	31	D	A										B	OE	138	Rc
addmex	31	D	A										0 0 0 0 0	OE	234	Rc
addzex	31	D	A										0 0 0 0 0	OE	202	Rc

divdx ⁴	31	D	A	B	OE	489	Rc
divdux ⁴	31	D	A	B	OE	457	Rc
divwx	31	D	A	B	OE	491	Rc
divwux	31	D	A	B	OE	459	Rc
mulhdx ⁴	31	D	A	B	0	73	Rc
mulhdux ⁴	31	D	A	B	0	9	Rc
mulhwx	31	D	A	B	0	75	Rc
mulhwux	31	D	A	B	0	11	Rc
mulldx ⁴	31	D	A	B	OE	233	Rc
mullwx	31	D	A	B	OE	235	Rc
negx	31	D	A	00000	OE	104	Rc
subfx	31	D	A	B	OE	40	Rc
subfcx	31	D	A	B	OE	8	Rc
subfex	31	D	A	B	OE	136	Rc
subfmex	31	D	A	00000	OE	232	Rc
subfzex	31	D	A	00000	OE	200	Rc

Table A-42. A-Form

OPCD	D	A	B	00000	XO	Rc
OPCD	D	A	B	C	XO	Rc
OPCD	D	A	00000	C	XO	Rc
OPCD	D	00000	B	00000	XO	Rc

Specific Instructions

Name	0	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
faddx	63	D	A	B	00000	21	Rc																					
faddSx	59	D	A	B	00000	21	Rc																					
fdivx	63	D	A	B	00000	18	Rc																					
fdivsX	59	D	A	B	00000	18	Rc																					
fmaddx	63	D	A	B	C	29	Rc																					
fmaddSx	59	D	A	B	C	29	Rc																					
fmsubx	63	D	A	B	C	28	Rc																					
fmsubSx	59	D	A	B	C	28	Rc																					
fmulx	63	D	A	00000	C	25	Rc																					
fmulSx	59	D	A	00000	C	25	Rc																					

Instructions Sorted by Form

fnmaddx ⁷	63	D	A	B	C	31	Rc
fnmaddSx	59	D	A	B	C	31	Rc
fnmsubx ⁷	63	D	A	B	C	30	Rc
fnmsubSx	59	D	A	B	C	30	Rc
fresx ^{5,7}	59	D	00000	B	00000	24	Rc
frsqrtox ^{5,7}	63	D	00000	B	00000	26	Rc
fselx ^{5,7}	63	D	A	B	C	23	Rc
fsqrtox ^{5,7}	63	D	00000	B	00000	22	Rc
fsqrtsx ^{5,7}	59	D	00000	B	00000	22	Rc
fsubx	63	D	A	B	00000	20	Rc
fsubsx	59	D	A	B	00000	20	Rc

Table A-43. M-Form

OPCD	S	A	SH	MB	ME	Rc
OPCD	S	A	B	MB	ME	Rc

Specific Instructions

Name 0 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

rlwimix	20	S	A	SH	MB	ME	Rc
rlwinmx	21	S	A	SH	MB	ME	Rc
rlwnmx	23	S	A	B	MB	ME	Rc

Table A-44. MD-Form

OPCD	S	A	sh	mb	XO	sh	Rc
OPCD	S	A	sh	me	XO	sh	Rc

Specific Instructions

Name 0 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

rldicx ⁴	30	S	A	sh	mb	2	sh	Rc
rldiclx ⁴	30	S	A	sh	mb	0	sh	Rc
rldicrx ⁴	30	S	A	sh	me	1	sh	Rc
rldimix ⁴	30	S	A	sh	mb	3	sh	Rc

Table A-45. MDS-Form

OPCD	S	A	B	mb	XO	Rc
OPCD	S	A	B	me	XO	Rc

Specific Instructions

Name	0	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
rdclx ⁴	30	S	A	B	mb	8	Rc																					
rdcrx ⁴	30	S	A	B	me	9	Rc																					

- ¹ Supervisor-level instruction
- ² Supervisor- and user-level instruction
- ³ Load and store string or multiple instruction
- ⁴ 64-bit instruction
- ⁵ Optional in the PowerPC architecture
- ⁶ MPC8240-implementation specific instruction

A.5 Instruction Set Legend

Table A-46 provides general information on the PowerPC instruction set (such as the architectural level, privilege level, and form).

Key:



Reserved bits



Instruction not implemented in the MPC8240

Table A-46. PowerPC Instruction Set Legend

	UISA	VEA	OEA	Supervisor Level	64-Bit	Optional	Form
addx	✓						XO
addcx	✓						XO
addex	✓						XO
addi	✓						D
addic	✓						D
addic.	✓						D
addis	✓						D
addmex	✓						XO
addzex	✓						XO
andx	✓						X
andcx	✓						X
andi.	✓						D
andis.	✓						D
bx	✓						I
bcx	✓						B
bcctrx	✓						XL
bclrx	✓						XL
cmp	✓						X
cmpi	✓						D
cmpl	✓						X
cmpli	✓						D
cntlzdx ⁴	✓				✓		X
cntlzwx	✓						X
crand	✓						XL
crandc	✓						XL
creqv	✓						XL

Instruction Set Legend

crnand	✓						XL
crnor	✓						XL
cror	✓						XL
crorc	✓						XL
crxor	✓						XL
dcbf		✓					X
dcbi ¹			✓	✓			X
dcbst		✓					X
dcbt		✓					X
dcbtst		✓					X
dcbz		✓					X
divdx ⁴	✓				✓		XO
divdux ⁴	✓				✓		XO
divwx	✓						XO
divwux	✓						XO
eciwx		✓				✓	X
ecowx		✓				✓	X
eieio		✓					X
eqvx	✓						X
extsbx	✓						X
extshx	✓						X
extswx ⁴	✓				✓		X
fabsx	✓						X
faddx	✓						A
faddSx	✓						A
fcfidx ^{4,7}	✓				✓		X
fcmpo ⁷	✓						X
fcmpt ⁷	✓						X
fctidx ^{4,7}	✓				✓		X
fctidzx ^{7,4}	✓				✓		X
fctiwx	✓						X
fctiwzx	✓						X
fdivx	✓						A
fdivsx	✓						A
fmaddx	✓						A

Instruction Set Legend

fmaddsx⁷	√						A
fmr_x	√						X
fmsub_x	√						A
fmsubsx⁷	√						A
fmul_x	√						A
fmuls_x	√						A
fnabsx⁷	√						X
fneg_x	√						X
fnmadd_x⁷	√						A
fnmaddsx	√						A
fnmsub_x	√						A
fnmsubsx	√						A
fres_x^{5,7}	√				√		A
frsp_x	√						X
frsqrte_x^{5,7}	√				√		A
fsel_x^{5,7}	√				√		A
fsqrt_x⁷	√				√		A
fsqrt_x^{5,7}	√				√		A
fsub_x	√						A
fsubsx	√						A
icbi		√					X
isync		√					XL
l_{bz}	√						D
l_{bzu}	√						D
l_{bzux}	√						X
l_{bzx}	√						X
l_d⁴	√				√		DS
l_{darx}⁴	√				√		X
l_{du}⁴	√				√		DS
l_{dux}⁴	√				√		X
l_{dx}⁴	√				√		X
l_{fd}⁷	√						D
l_{fd_u}⁷	√						D
l_{fd_{ux}}⁷	√						X
l_{fd_x}⁷	√						X

Instruction Set Legend

	UISA	VEA	OEA	Supervisor Level	64-bit	Optional	Form
lfs ⁷	✓						D
lfsu ⁷	✓						D
lfsux ⁷	✓						X
lfsx ⁷	✓						X
lha	✓						D
lhau	✓						D
lhaux	✓						X
lhax	✓						X
lhbrx	✓						X
lhz	✓						D
lhzu	✓						D
lhzux	✓						X
lhzx	✓						X
lmw ³	✓						D
lswi ³	✓						X
lswx ³	✓						X
lwa ⁴	✓				✓		DS
lwarx	✓						X
lwau ⁴	✓				✓		X
lwax ⁴	✓				✓		X
lwbrx	✓						X
lwz	✓						D
lwzu	✓						D
lwzux	✓						X
lwzx	✓						X
mcrf	✓						XL
mcrfs ⁷	✓						X
mcrxr	✓						X
mfcrr	✓						X
mffsx	✓						X
mfmsr ¹			✓	✓			X
mfspr ²	✓		✓	✓			XFX
mfsr ¹			✓	✓			X
mfsrin ¹			✓	✓			X

Instruction Set Legend

	UISA	VEA	OEA	Supervisor Level	64-bit	Optional	Form
mtfb		√					AFX
mtrcf	√						AFX
mtfsb0x	√						X
mtfsb1x ⁷	√						X
mtfsfx	√						XFL
mtfsfix	√						X
mtmsr ¹			√	√			X
mtspr ²	√		√	√			AFX
mtsr ¹			√	√			X
mtsrin ¹			√	√			X
mulhdx ⁴	√				√		XO
mulhdux ⁴	√				√		XO
mulhwx	√						XO
mulhwux	√						XO
mulldx ⁴	√				√		XO
mulli	√						D
mullwx	√						XO
nandx	√						X
negx	√						XO
norx	√						X
orx	√						X
orcx	√						X
ori	√						D
oris	√						D
rfi ¹			√	√			XL
ridclx ⁴	√				√		MDS
ridcrx ⁴	√				√		MDS
rdicx ⁴	√				√		MD
rdiclx ⁴	√				√		MD
rdicrx ⁴	√				√		MD
rdimix ⁴	√				√		MD
rlwimix	√						M
rlwinmx	√						M
rlwnmx	√						M

Instruction Set Legend

	UISA	VEA	OEA	Supervisor Level	64-bit	Optional	Form
sc	√		√				SC
slbia ^{1,4,5}			√	√	√	√	X
sible ^{1,4,5}			√	√	√	√	X
sldx ⁴	√				√		X
slwx	√						X
sradx ⁴	√				√		X
sradix ⁴	√				√		XS
srawx	√						X
srawix	√						X
srdx ⁴	√				√		X
srwx	√						X
stb	√						D
stbu	√						D
stbux	√						X
stbx	√						X
std ⁴	√				√		DS
stdcx ⁴	√				√		X
stdu ⁴	√				√		DS
stdux ⁴	√				√		X
stdx ⁴	√				√		X
stfd ⁷	√						D
stfdu ⁷	√						D
stfdx ⁷	√						X
stfdx ⁷	√						X
stfiwx ^{5,7}	√					√	X
stfs ⁷	√						D
stfsu ⁷	√						D
stfsux ⁷	√						X
stfsx ⁷	√						X
sth	√						D
sthbrx	√						X
sthu	√						D
sthux	√						X
sthx	√						X

Instruction Set Legend

stmw ³	√						D
stswi ³	√						X
stswx ³	√						X
stw	√						D
stwbrx	√						X
stwcx.	√						X
stwu	√						D
stwux	√						X
stwx	√						X
subfx	√						XO
subfcx	√						XO
subfex	√						XO
subfic	√						D
subfmex	√						XO
subfzex	√						XO
sync	√						X
td ⁴	√				√		X
tdi ⁴	√				√		D
tlbia ^{1,5}			√	√		√	X
tlbie ^{1,5}			√	√		√	X
tlbld ^{1,6}				√			X
tlbli ^{1,6}				√			X
tlbsync ^{1,5}			√	√			X
tw	√						X
twi	√						D
xorx	√						X
xori	√						D
xoris	√						D

¹ Supervisor-level instruction

² Supervisor- and user-level instruction

³ Load and store string or multiple instruction

⁴ 64-bit instruction

⁵ Optional in the PowerPC architecture

⁶ MPC8240-implementation specific instruction

Appendix B

Bit and Byte Ordering

The MPC8240 supports two byte-ordering conventions for the PCI bus and local memory—big-endian and little-endian. This appendix describes both modes and provides examples of each. Chapter 3, “Operand Conventions,” in *PowerPC Microprocessor Family: The Programming Environments for 32-bit Microprocessors*, provides a general overview of byte ordering and describes byte ordering for PowerPC microprocessors.

B.1 Byte Ordering Overview

For big-endian data, the MSB is stored at the lowest (or starting) address and the LSB is stored at the highest (or ending) address. This is called big-endian because the big (most-significant) end of the scalar comes first in memory.

For true little-endian byte ordering, the LSB is stored at the lowest address while the MSB is stored at the highest address. This is called true little-endian because the little (least-significant) end of the scalar comes first in memory.

For PowerPC little-endian byte ordering (also referred to as munged little-endian), the address of data is modified so that the memory structure appears to be little-endian to the executing processor, when, in fact, the byte ordering is big-endian. The address modification is called munging. Note that the term ‘munging’ is not defined or used in the PowerPC architecture specification, but is commonly used to describe address modifications. The byte ordering is called PowerPC little-endian because PowerPC microprocessors use this method to operate in little-endian mode.

In addition, the PCI bus uses a bit format where the most-significant bit (msb) for data is AD31, while the processor core and the internal peripheral logic data bus use a bit format where the msb is DH0. Thus, PCI data bit AD31 equates to the processor’s data bits DH0 and DL0, while PCI data bit AD0 equates to the processor’s data bits DH31 and DL31.

B.2 Byte-Ordering Mechanisms

Several byte-ordering mechanisms in the MPC8240 are controlled by programmable parameters. The PowerPC architecture defines two bits in the MSR for specifying byte ordering—LE (little-endian mode) and ILE (exception little-endian mode). For the MPC8240 these bits control only the addresses generated by the PowerPC core. The LE bit

Big-Endian Mode

specifies the endian mode for normal core operation and ILE specifies the mode to be used when an exception handler is invoked. That is, when an exception occurs, the ILE bit (as set for the interrupted process) is copied into MSR[LE] to select the endian mode for the context established by the exception. The LE and ILE bits control a 3-bit address modifier in the processor core.

To convert from PowerPC little-endian to true little-endian byte ordering, all the byte lanes must be reversed (MSB to LSB, etc.) and the addresses must be unmunged external to the processor core. When configured for little-endian mode, the MPC8240 unmunges the address and reverses the byte lanes between the PCI bus and local memory in the central control unit (CCU). This means that the data in local memory is stored using PowerPC little-endian byte ordering, but data on the PCI bus is in true little-endian byte order. The PICR1[LE_MODE] parameter controls a 3-bit address modifier and byte lane swapper in the CCU.

Note that the processor core and the CCU should be set for the same endian mode before accessing devices on the PCI bus.

B.3 Big-Endian Mode

When the processor core is operating in big-endian mode, no address modification is performed by the processor. In big-endian mode, the MPC8240 maintains the big-endian byte ordering on the PCI bus during the data phase(s) of PCI transactions. The byte lane translation for big-endian mode is shown in Table B-1. Note that the bit ordering on the PCI bus remains unchanged (that is, AD31 is still the msb of the byte in byte lane 3 and AD0 is still the lsb of the byte in byte lane 0).

Table B-1. Byte Lane Translation in Big-Endian Mode

Processor Byte Lane	Processor Data Bus Signals	PCI Byte Lane	PCI Address/Data Bus Signals During PCI Data Phase
0	DH[0–7]	0	AD[7–0]
1	DH[8–15]	1	AD[15–8]
2	DH[16–23]	2	AD[23–16]
3	DH[24–31]	3	AD[31–24]
4	DL[0–7]	0	AD[7–0]
5	DL[8–15]	1	AD[15–8]
6	DL[16–23]	2	AD[23–16]
7	DL[24–31]	3	AD[31–24]

Figure B-1 shows a 4-byte write to PCI memory space in big-endian mode.

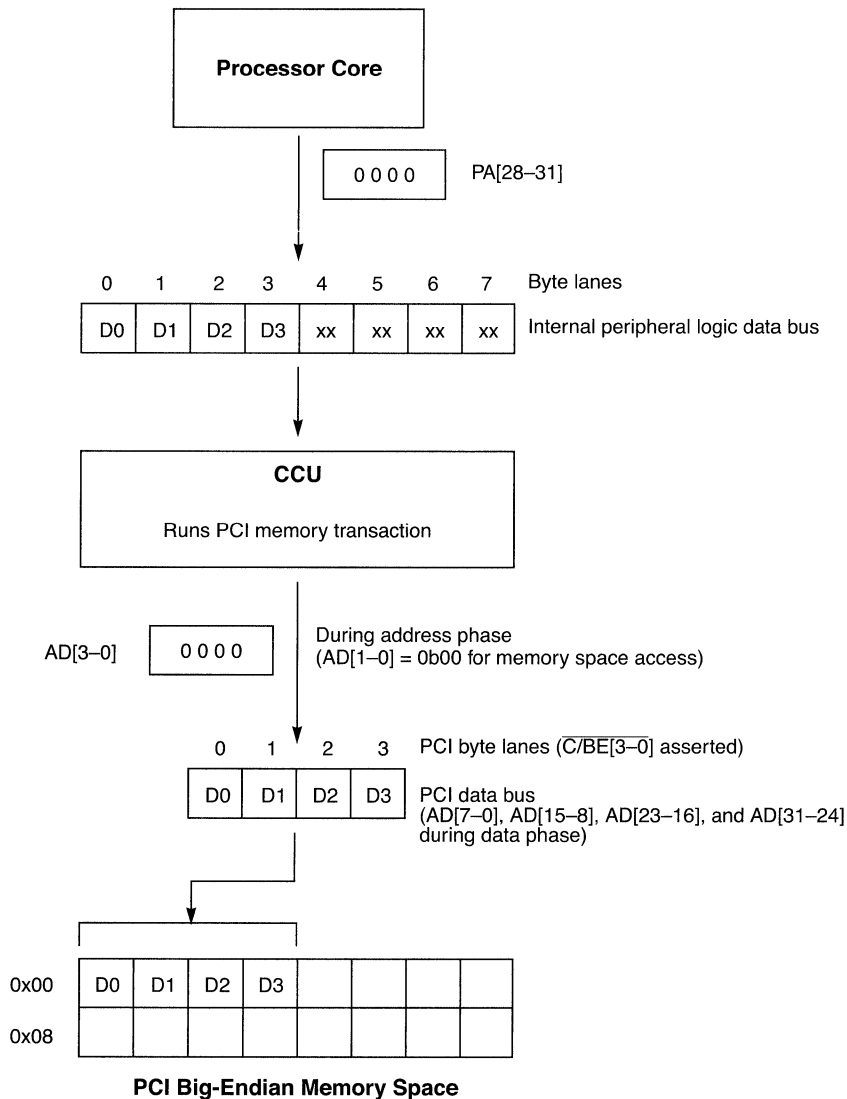


Figure B-1. Four-Byte Transfer to PCI Memory Space—Big-Endian Mode

Note that the MSB on the internal peripheral logic bus, D0, is placed on byte lane 0 (AD[7-0]) on the PCI bus. This occurs so D0 appears at address 0xnnnn_nn00 and not at address 0xnnnn_nn03 in the PCI space.

Big-Endian Mode

The following example demonstrates the operation of a system in big-endian mode. Starting with a program that does the following:

```
store string ("hello, world") at 0x000
store pointer (0xFEDCBA98) at 0x010
store halfword (0d1234) at 0x00E
store byte (0x55) at 0x00D
```

If the data is stored into local memory, it appears as shown in Figure B-2.

Contents	'h'	'e'	'l'	'l'	'o'	','	' '	'w'
Address	00	01	02	03	04	05	06	07
Contents	'o'	'r'	'l'	'd'		0x55	12	34
Address	08	09	0A	0B	0C	0D	0E	0F
Contents	0xFE	0xDC	0xBA	0x98				
Address	10	11	12	13	14	15	16	17

Figure B-2. Big-Endian Memory Image in Local Memory

Note that the stored data has big-endian ordering. The 'h' is at address 0x000.

If the data is stored to the PCI memory space, it appears as shown in Figure B-3.

Contents	'l'	'l'	'e'	'h'
Address	03	02	01	00

Contents	'w'	' '	'.'	'o'
Address	07	06	05	04

Contents	'd'	'l'	'r'	'o'
Address	0B	0A	09	08

Contents	34	12	0x55	0x00
Address	0F	0E	0D	0C

Contents	0x98	0xBA	0xDC	0xFE
Address	13	12	11	10

Contents				
Address	17	16	15	14

Figure B-3. Big-Endian Memory Image in Big-Endian PCI Memory Space

Note that the string 'hello, world' starts at address 0x000. The other data is stored to the desired location with big-endian byte ordering.

B.4 Little-Endian Mode

When the processor core is operating in little-endian mode, its internal BIU modifies each address. This modification is called munging. The processor munges the address by exclusive-ORing the three low-order address bits with a three-bit value that depends on the length of the operand (1, 2, 4, or 8 bytes), as shown in Table B-2.

Table B-2. Processor Address Modification for Individual Aligned Scalars

Data Length (in Bytes)	Address Modification A[29–31]
8	No change
4	XOR with 0b100
2	XOR with 0b110
1	XOR with 0b111

Note that munging makes it appear to the processor that individually aligned scalars are stored in little-endian byte order, when in fact, they are stored in big-endian order, but at different byte addresses within double words. Only the address is modified, not the byte order.

The munged address is used by the memory interface of the MPC8240 to access local memory. To provide true little-endian byte-ordering to the PCI bus, the MPC8240 unmunges the address to its original value and the byte lanes are reversed. The MPC8240 unmunges aligned addresses by exclusive-ORing the three low-order address bits with a three-bit value that depends on the length of the operand (1, 2, 3, 4, or 8 bytes), as shown in Table B-3.

Table B-3. MPC8240 Address Modification for Individual Aligned Scalars

Data Length (in Bytes)	Address Modification A[29–31]
8	No change
4	XOR with 0b100
3	XOR with 0b101
2	XOR with 0b110
1	XOR with 0b111

The MPC8240 also supports misaligned 2-byte transfers that do not cross word boundaries in little-endian mode. The MPC8240 exclusive-ORs the address with 0x100. Note that the MPC8240 does not support 2-byte transfers that cross word boundaries in little-endian mode.

The byte lane translation for little-endian mode is shown in Table B-4.

Table B-4. Byte Lane Translation in Little-Endian Mode

Processor Byte Lane	Processor Data Bus Signals	PCI Byte Lane	PCI Address/Data Bus Signals During PCI Data Phase
0	DH[0–7]	3	AD[31–24]
1	DH[8–15]	2	AD[23–16]
2	DH[16–23]	1	AD[15–8]
3	DH[24–31]	0	AD[7–0]
4	DL[0–7]	3	AD[31–24]
5	DL[8–15]	2	AD[23–16]
6	DL[16–23]	1	AD[15–8]
7	DL[24–31]	0	AD[7–0]

Starting with the same program as before:

```
store string ("hello, world") at 0x000
store pointer (0xFEDCBA98) at 0x010
store halfword (0d1234) at 0x00E
store byte (0x55) at 0x00D
```

If the data is stored to local memory in little-endian mode, the MPC8240 stores the data to the munged addresses in local memory as shown in Figure B-4.

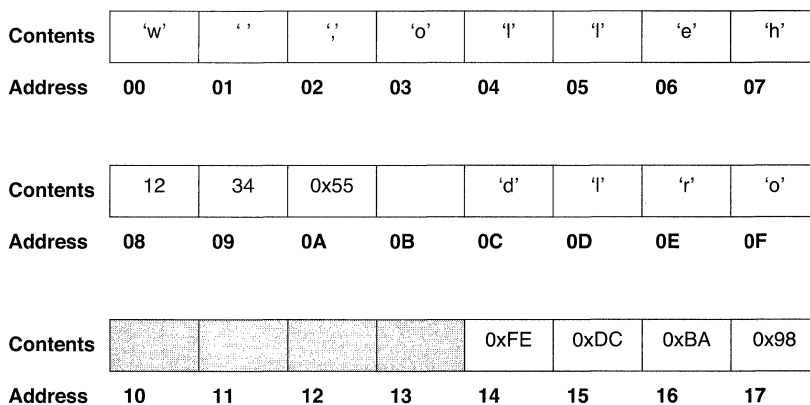


Figure B-4. Munged Memory Image in local memory

The image shown in Figure B-4 is not a true little-endian mapping. Note how munging has changed the addresses of the data. The 'h' is now at address 0x007. Also note that the byte

Little-Endian Mode

ordering of the word-length pointer, 0xFEDCBA98, is big-endian; only the address has been modified. However, since the processor core munges the address when accessing local memory, the mapping appears little-endian to the processor core.

If the data is stored to the PCI memory space, or if a PCI agent reads from local memory, the MPC8240 unmunges the addresses and reverses the byte-ordering before sending the data out to the PCI bus. The data is stored to little-endian PCI memory space as shown in Figure B-5.

Contents	'l'	'l'	'e'	'h'
Address	03	02	01	00
Contents	'w'	' '	','	'o'
Address	07	06	05	04
Contents	'd'	'l'	'r'	'o'
Address	0B	0A	09	08
Contents	12	34	0x55	0x00
Address	0F	0E	0D	0C
Contents	0xFE	0xDC	0xBA	0x98
Address	13	12	11	10
Contents				
Address	17	16	15	14

Figure B-5. Little-Endian Memory Image in Little-Endian PCI Memory Space

Note that the string 'hello, world' starts at address 0x000. The other data is stored to the desired location with true little-endian byte ordering.

Figure B-6 through Figure B-11 show the munging/unmunging process for transfers to the PCI memory space and to the PCI I/O space.

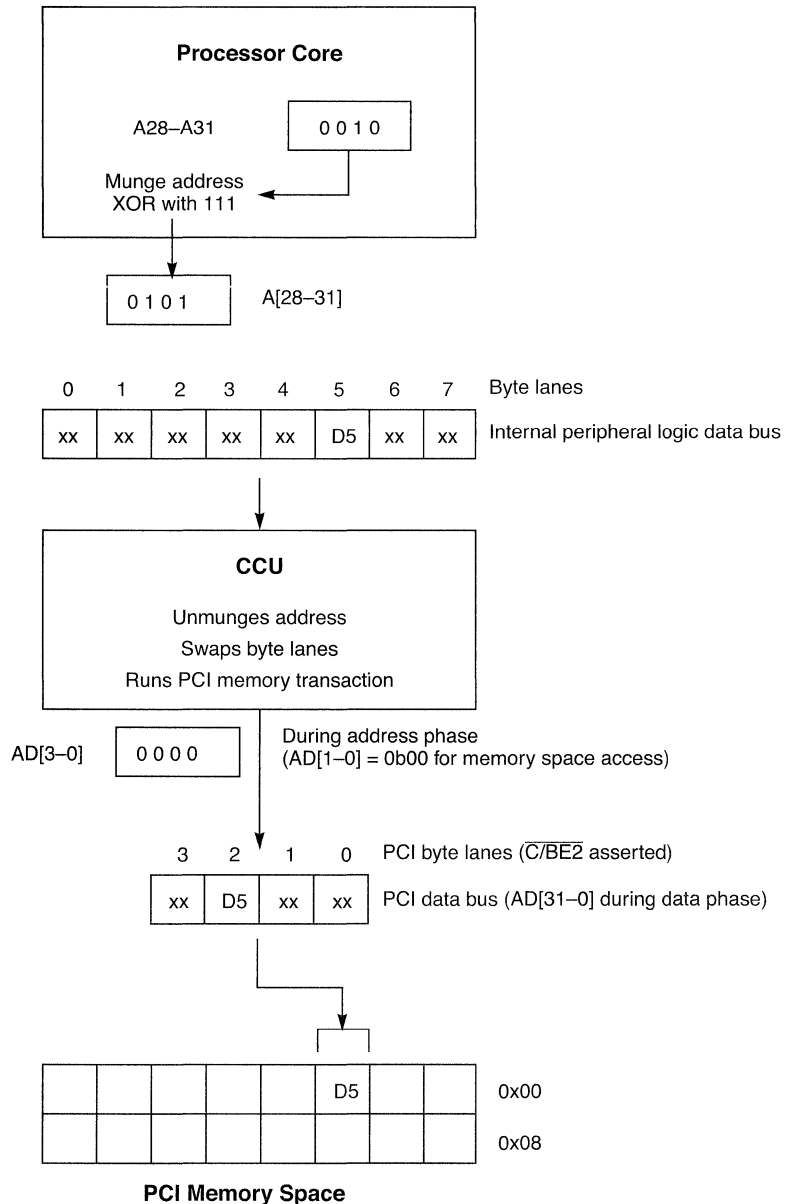


Figure B-6. One-Byte Transfer to PCI Memory Space—Little-Endian Mode

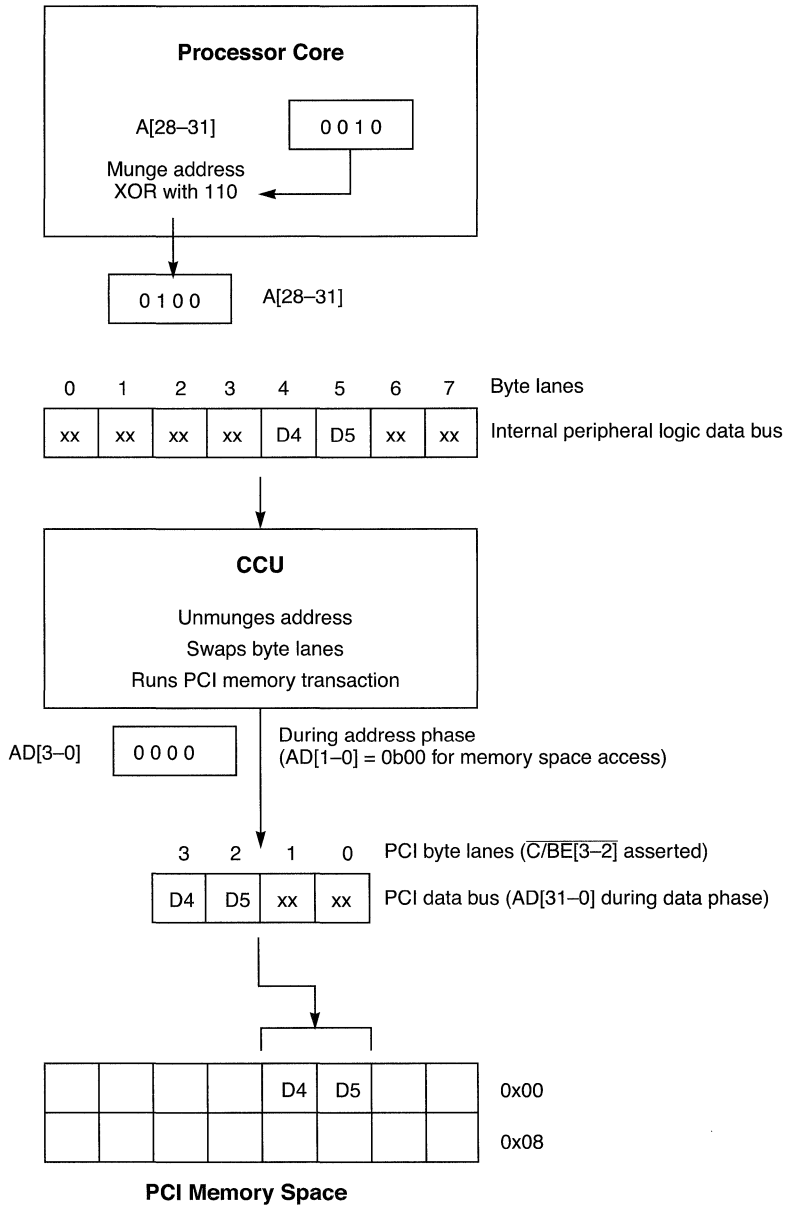


Figure B-7. Two-Byte Transfer to PCI Memory Space—Little-Endian Mode

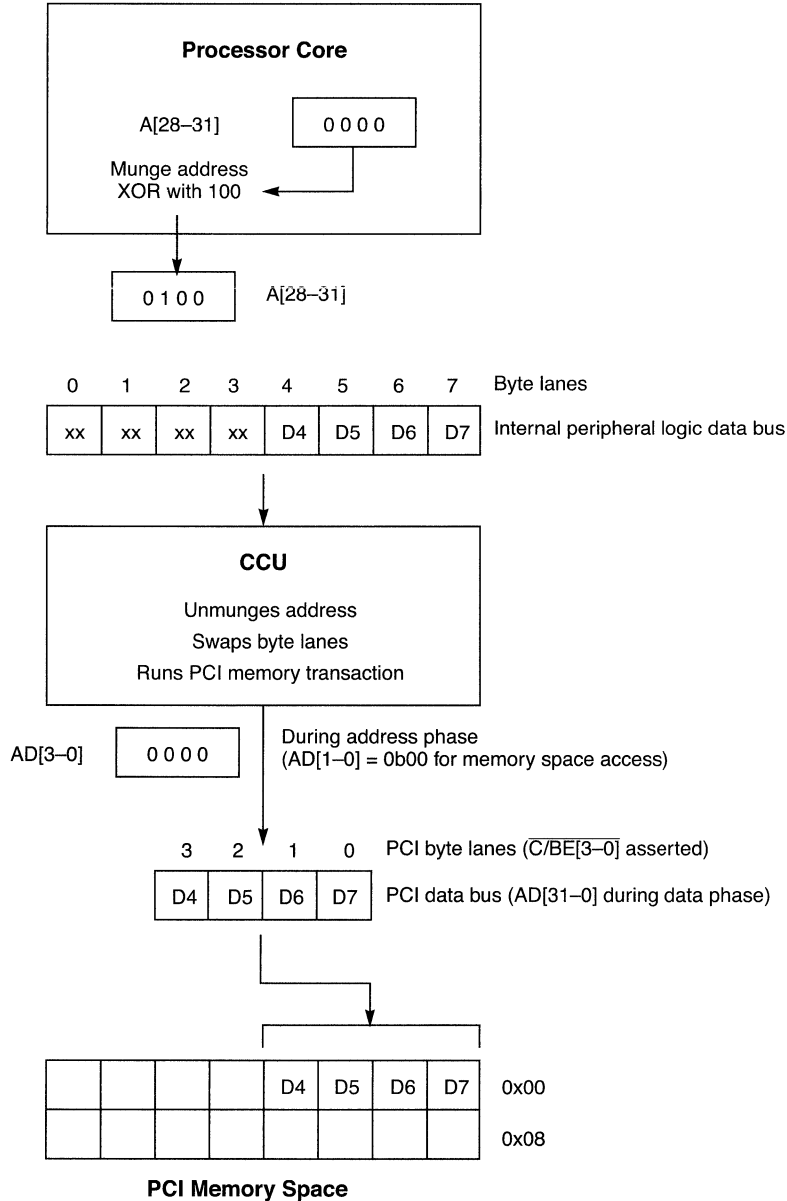


Figure B-8. Four-Byte Transfer to PCI Memory Space—Little-Endian Mode

Little-Endian Mode

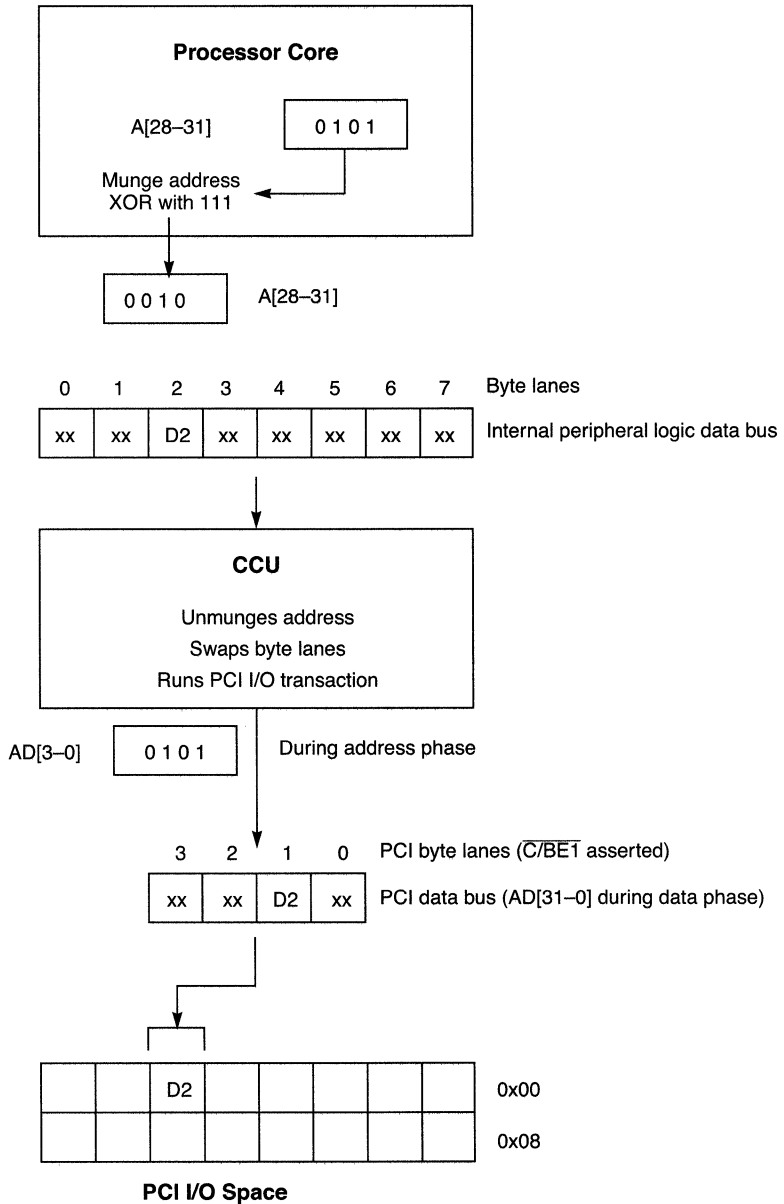


Figure B-9. One-Byte Transfer to PCI I/O Space—Little-Endian Mode

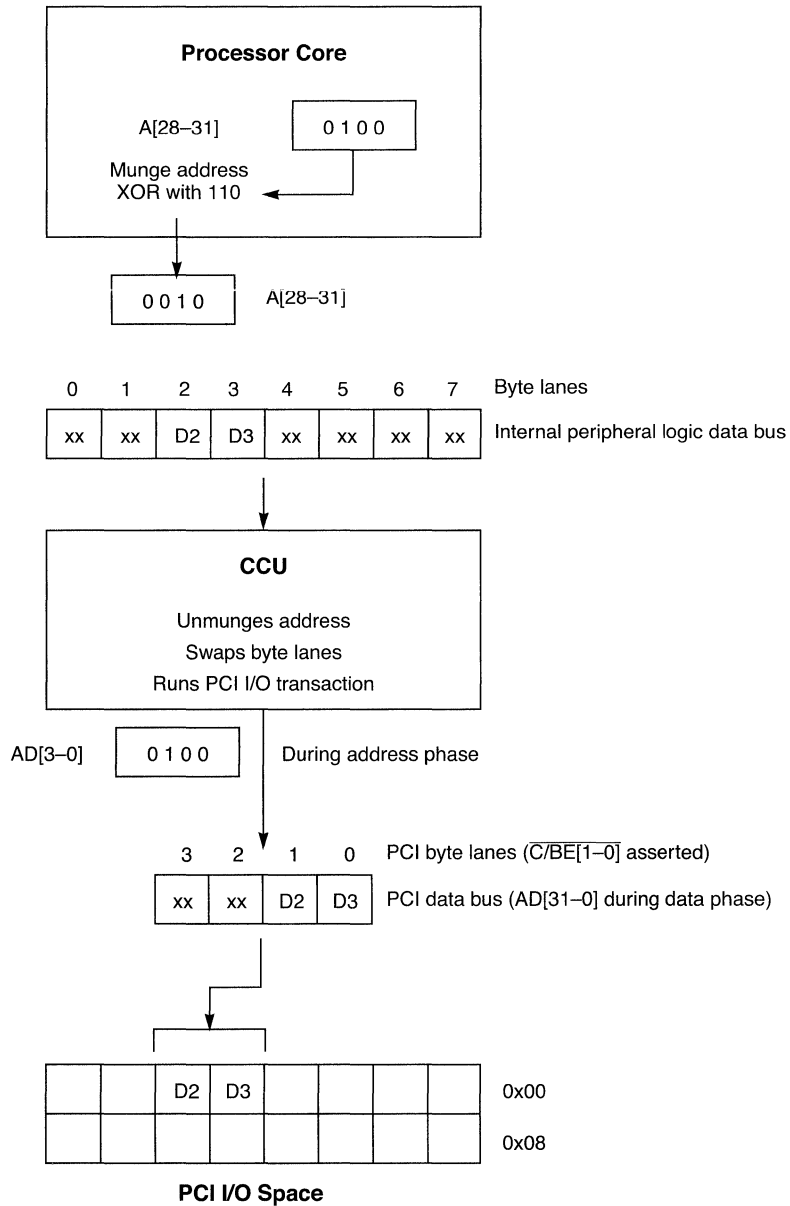


Figure B-10. Two-Byte Transfer to PCI I/O Space—Little-Endian Mode

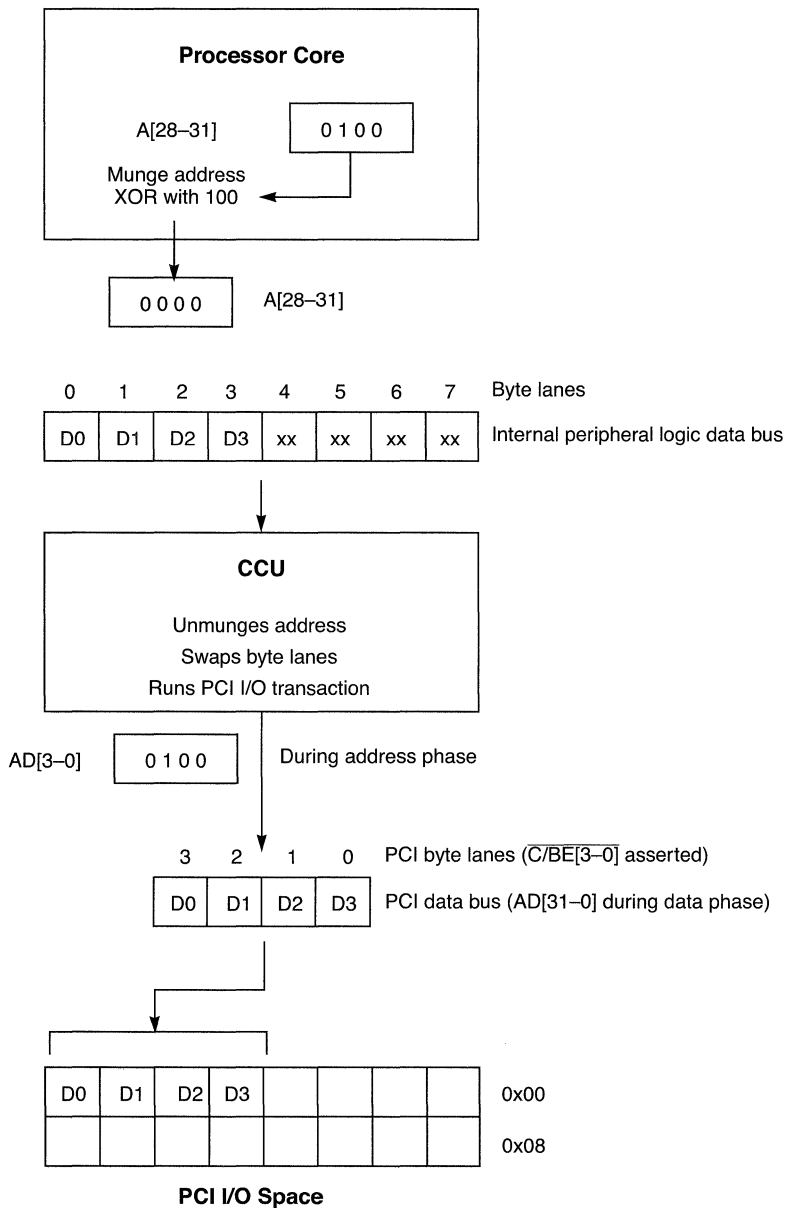


Figure B-11. Four-Byte Transfer to PCI I/O Space— Little-Endian Mode

B.4.1 I/O Addressing in Little-Endian Mode

For a system running in big-endian mode, both the processor core and the memory subsystem recognize the same byte as byte 0. However, this is not true for a system running in little-endian mode because of the munged address bits when the MPC8240 accesses external memory.

For I/O transfers in little-endian mode to transfer bytes properly, they must be performed as if the bytes transferred were accessed one at a time, using the little-endian address modification appropriate for the single-byte transfers (that is, the lowest order address bits must be XORed with 0b111). This does not mean that I/O operations in little-endian systems must be performed using only one-byte-wide transfers. Data transfers can be as wide as desired, but the order of the bytes within double words must be as if they were fetched or stored one at a time. That is, for a true little-endian I/O device, the system must provide a way to munge and unmunge the addresses and reverse the bytes within a doubleword (MSB to LSB).

A load or store that maps to a control register on an external device may require the bytes of the register data to be reversed. If this reversal is required, the load and store with byte-reverse instructions (**lhbrx**, **lwbrx**, **sthbrx**, and **stwbrx**) may be used.

B.5 Setting the Endian Mode Of Operation

The MPC8240 powers up in big-endian mode. The endian mode should be set early in the initialization routine and remain unchanged for the duration of system operation. To switch between the different endian modes of operation, the core must run in serialized mode and the caches should be disabled. Switching back and forth between modes is not recommended.

To switch the system from big-endian to little-endian mode, the MSR[LE] and MSR[ILE] bits should be set using an **mtmsr** instruction that resides on an odd word boundary ($A[29] = 1$). The instruction that is executed next will be fetched from this address plus 8. If the **mtmsr** instruction resides on an even word boundary ($A[29] = 0$), then the instruction will be executed twice due to the address munging of little-endian mode. After the processor core has been programmed for little-endian mode, the LE_MODE parameter in the peripheral control logic should be set.

To switch the system from little- to big-endian mode, the MSR[LE] and MSR[ILE] bits should be cleared using an **mtmsr** instruction that resides on an even word boundary ($A[29] = 0$). The instruction that is executed next is fetched from this address plus 12. After the processor core is programmed for big-endian mode, the LE_MODE parameter in the peripheral control logic should be cleared.

Glossary of Terms and Abbreviations

The glossary contains an alphabetical list of terms, phrases, and abbreviations used in this book. Some of the terms and definitions included in the glossary are reprinted from IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic, copyright ©1985 by the Institute of Electrical and Electronics Engineers, Inc. with the permission of the IEEE.

A **Atomic.** A bus access that attempts to be part of a read-write operation to the same address uninterrupted by any other access to that address (the term refers to the fact that the transactions are indivisible). The PowerPC 603e microprocessor initiates the read and write separately, but signals the memory system that it is attempting an atomic operation. If the operation fails, status is kept so that the 603e can try again. The 603e implements atomic accesses through the **lwarx/stwcx** instruction pair.

B **Beat.** A single state on the 603e bus interface that may extend across multiple bus cycles. A 603e transaction can be composed of multiple address or data *beats*.

Biased exponent. The sum of the exponent and a constant (bias) chosen to make the biased exponent's range non-negative.

Big-endian. A byte-ordering method in memory where the address *n* of a word corresponds to the most significant byte. In an addressed memory word, the bytes are ordered (left to right) 0, 1, 2, 3, with 0 being the most significant byte.

Boundedly undefined. The results of attempting to execute a given instruction are said to be *boundedly undefined* if they could have been achieved by executing an arbitrary sequence of defined instructions, in valid form, starting in the state the machine was in before attempting to execute the given instruction. Boundedly undefined results for a given instruction may vary between implementations, and between execution attempts in the same implementation.

Branch folding. A technique of removing the branch instruction from the instruction sequence.

Burst. A multiple beat data transfer whose total size is typically equal to a cache block (in the 603e, a 32-byte block).

Bus clock. Clock that causes the bus state transitions.

Bus master. The owner of the address or data bus; the device that initiates or requests the transaction.

C

Cache. High-speed memory containing recently accessed data and/or instructions (subset of main memory).

Cache block. The cacheable unit for a PowerPC processor. The size of a cache block may vary among processors. For the 603e, it is one cache line (8 words).

Cache coherency. Caches are coherent if a processor performing a read from its cache is supplied with data corresponding to the most recent value written to memory or to another processor's cache.

Cast-outs. Cache block that must be written to memory when a snoop miss causes the least recently used block with modified data to be replaced.

Context synchronization. Context synchronization is the result of specific instructions (such as **sc** or **rfi**) or when certain events occur (such as an exception). During context synchronization, all instructions in execution complete past the point where they can produce an exception; all instructions in execution complete in the context in which they began execution; all subsequent instructions are fetched and executed in the new context.

Copy-back operation. A cache operation in which a cache line is copied back to memory to enforce cache coherency. Copy-back operations consist of snoop push-out operations and cache cast-out operations.

D

Denormalized number. A nonzero floating-point number whose exponent has a reserved value, usually the format's minimum, and whose explicit or implicit leading significand bit is zero.

Direct-store segment access. An access to an I/O address space. The 603 defines separate memory-mapped and I/O address spaces, or segments, distinguished by the corresponding segment register T bit in the address translation logic of the 603. If the T bit is cleared, the memory reference is a normal memory-mapped access and can use the virtual memory management hardware of the 603. If the T bit is set, the memory reference is a direct-store access.

E **Exception.** An unusual or error condition encountered by the processor that results in special processing.

Exception handler. A software routine that executes when an exception occurs. Normally, the exception handler corrects the condition that caused the exception, or performs some other meaningful task (such as aborting the program that caused the exception). The addresses of the exception handlers are defined by a two-word exception vector that is branched to automatically when an exception occurs.

Exclusive state. EMI state (E) in which only one caching device contains data that is also in system memory.

Execution synchronization. All instructions in execution are architecturally complete before beginning execution (appearing to begin execution) of the next instruction. Similar to context synchronization but doesn't force the contents of the instruction buffers to be deleted and refetched.

Exponent. The component of a binary floating-point number that normally signifies the integer power to which two is raised in determining the value of the represented number. Occasionally the exponent is called the signed or unbiased exponent.

F **Feed-forwarding.** A 603e feature that reduces the number of clock cycles that an execution unit must wait to use a register. When the source register of the current instruction is the same as the destination register of the previous instruction, the result of the previous instruction is routed to the current instruction at the same time that it is written to the register file. With feed-forwarding, the destination bus is gated to the waiting execution unit over the appropriate source bus, saving the cycles which would be used for the write and read.

Floating-point unit. The functional unit in the 603e processor responsible for executing all floating-point instructions.
(Not supported on the EC603e microprocessor)

Flush. An operation that causes a modified cache block to be invalidated and the data to be written to memory.

Fraction. The field of the significand that lies to the right of its implied binary point.

G

General-purpose register. Any of the 32 registers in the 603e register file. These registers provide the source operands and destination results for all 603e data manipulation instructions. Load instructions move data from memory to registers, and store instructions move data from registers to memory.

I

IEEE 754. A standard written by the Institute of Electrical and Electronics Engineers that defines operations of binary floating-point arithmetic and representations of binary floating-point numbers.

Instruction queue. A holding place for instructions fetched from the current instruction stream.

Integer unit. The functional unit in the 603e responsible for executing all integer instructions.

Interrupt. An external signal that causes the 603e to suspend current execution and take a predefined exception.

Invalid state. EMI state (I) that indicates that the cache block does not contain valid data.

K

Kill. An operation that causes a cache block to be invalidated.

L

Latency. The number of clock cycles necessary to execute an instruction and make ready the results of that instruction.

Little-endian. A byte-ordering method in memory where the address *n* of a word corresponds to the least significant byte. In an addressed memory word, the bytes are ordered (left to right) 3, 2, 1, 0, with 3 being the most significant byte.

Livelock. A state in which processors interact in a way such that no processor makes progress.

M

Mantissa. The decimal part of logarithm.

Memory-mapped accesses. Accesses whose addresses use the segmented or block address translation mechanisms provided by the MMU and that occur externally with the bus protocol defined for memory.

Memory coherency. Refers to memory agreement between caches and system memory (for example, EMI cache coherency).

Memory consistency. Refers to levels of memory with respect to a single processor and system memory (for example, on-chip cache, secondary cache, and system memory).

Memory-forced I/O controller interface access. These accesses are made to memory space. They do not use the extensions to the memory protocol described for I/O controller interface accesses, and they bypass the page- and block-translation and protection mechanisms.

Memory management unit. The functional unit in the 603e that translates the logical address bits to physical address bits.

Modified state. EMI state (M) in which one, and only one, caching device has the valid data for that address. The data at this address in external memory is not valid.

N

NaN. An abbreviation for not a number; a symbolic entity encoded in floating-point format. There are two types of NaNs—signaling NaNs and quiet NaNs.

No-op. No-operation. A single-cycle operation that does not affect registers or generate bus activity.

O

Out-of-order. An operation is said to be out-of-order when it is not guaranteed to be required by the sequential execution model, such as the execution of an instruction that follows another instruction that may alter the instruction flow. For example, execution of instructions in an unresolved branch is said to be out-of-order, as is the execution of an instruction behind another instruction that may yet cause an exception. The results of operations that are performed out-of-order are not committed to architected resources until it can be ensured that these results adhere to the in-order, or sequential execution model.

Overflow. An error condition that occurs during arithmetic operations when the result cannot be stored accurately in the destination register(s). For example, if two 32-bit numbers are added, the sum may require 33 bits due to carry. Since the 32-bit registers of the 603e cannot represent this sum, an overflow condition occurs.

P

Packet. A term used in the 603 with respect to direct store operations.

Page. A 4-Kbyte area of memory, aligned on a 4-Kbyte boundary.

Park. The act of allowing a bus master to maintain mastership of the bus without having to arbitrate.

Pipelining. A technique that breaks instruction execution into distinct steps so that multiple steps can be performed at the same time.

Precise exceptions. The pipeline can be stopped so the instructions that preceded the faulting instruction can complete, and subsequent instructions can be executed following the execution of the exception handler. The system is precise unless one of the imprecise modes for invoking the floating-point enabled exception is in effect.

Q

Quiesce. To come to rest. The processor is said to quiesce when an exception is taken or a **sync** instruction is executed. The instruction stream is stopped at the decode stage and executing instructions are allowed to complete to create a controlled context for instructions that may be affected by out-of-order, parallel execution. See **Context synchronization**.

Quiet NaNs. Propagate through almost every arithmetic operation without signaling exceptions. These are used to represent the results of certain invalid operations, such as invalid arithmetic operations on infinities or on NaNs, when invalid.

S

Scan interface. The 603e's test interface.

Shadowing. Shadowing allows a register to be updated by instructions that are executed out of order without destroying machine state information.

Signaling NaNs. Signal the invalid operation exception when they are specified as arithmetic operands

Significand. The component of a binary floating-point number that consists of an explicit or implicit leading bit to the left of its implied binary point and a fraction field to the right.

Slave. The device addressed by a master device. The slave is identified in the address tenure and is responsible for supplying or latching the requested data for the master during the data tenure.

Snooping. Monitoring addresses driven by a bus master to detect the need for coherency actions.

Snoop push. Write-backs due to a snoop hit. The block will transition to an invalid or exclusive state.

Split-transaction. A transaction with independent request and response tenures.

Split-transaction Bus. A bus that allows address and data transactions from different processors to occur independently.

Static branch prediction. Mechanism by which software (for example, compilers) can give a hint to the machine hardware about the direction the branch is likely to take.

Superscalar machine. A machine that can issue multiple instructions concurrently from a conventional linear instruction stream.

Supervisor mode. The privileged operation state of the 603e. In supervisor mode, software can access all control registers and can access the supervisor memory space, among other privileged operations.

T

Tenure. The period of bus mastership. For the 603e, there can be separate address bus tenures and data bus tenures. A tenure consists of three phases: arbitration, transfer, termination

Transaction. A complete exchange between two bus devices. A transaction is minimally comprised of an address tenure; one or more data tenures may be involved in the exchange. There are two kinds of transactions: address/data and address-only.

Transfer termination. Signal that refers to both signals that acknowledge the transfer of individual beats (of both single-beat transfer and individual beats of a burst transfer) and to signals that mark the end of the tenure.

U

Underflow. An error condition that occurs during arithmetic operations when the result cannot be represented accurately in the destination register. For example, underflow can happen if two floating-point fractions are multiplied and the result is a single-precision number. The result may require a larger exponent and/or mantissa than the single-precision format makes available. In other words, the result is too small to be represented accurately.

User mode. The unprivileged operating state of the 603e. In user mode, software can only access certain control registers and can only access user memory space. No privileged operations can be performed.

W

Write-through. A memory update policy in which all processor write cycles are written to both the cache and memory.

INDEX

Numerics

603e core, *see* Processor core

A

Abbreviations, xxxviii

Acronyms, xxxviii

Address maps

address translation, 4-10

addressing on PCI bus, 8-9

debug address maps, 15-5

DMA controller interactions, 9-7

embedded utilities memory block, 4-16

emulation mode address map, 5-33

EPIC controller, 12-3

ESCR1 register, 5-33

examples, configuration sequences, 5-2

map A

overview, 4-1

registers, 5-1

map B

overview, 4-5

registers, 5-2

overview, 4-1

Addressing

address translation, 4-10

address translation registers, 4-12

embedded utilities memory block, 4-16

inbound PCI address translation, 4-10

outbound PCI address translation, 4-11

PCI bus

configuration space, 8-10

I/O space, 8-10

memory space, 8-10

addressing, 8-10

Addressing modes, 2-18

AD_n (PCI address/data bus) signals, 3-8, 8-10

Alignment

byte alignment, 8-11, B-2

Arbitration

I²C arbitration procedure, 11-5

internal arbitration, 7-9

PCI bus arbitration, 8-4

AR_n (ROM address) signal, 3-20

\overline{AS} (address strobe) signal, 3-22

B

BCR (byte count register), 9-18

Big-endian mode

accessing configuration registers, 5-3

byte lane translation, B-2

byte ordering, B-2

DMA descriptors, 9-11

LE_MODE bit, 5-23

PCI memory space, B-3, B-5

BIST (built-in self test) control register, 5-9

Block diagrams

clock subsystem block diagram, 3-32

DMA controller, 9-2

EDO DRAM interface, 6-6

EPIC controller, 12-3

EPIC controller internal block diagram, 12-8

FPM interface, 6-6

memory interface, 6-3

MPC8240 functional block diagram, 1-2

MPC8240 integrated processor core, 1-9

peripheral logic block diagram, 1-11

Port X interface, 6-72

processor core, 2-2

ROM interface, 6-61

SDRAM interface, 6-31

Boundary-scan registers, 15-23

Branch instructions

branch instructions, A-24

condition register logical, A-24

system linkage, A-24

trap, A-24

Buffers

internal buffers

copy-back buffer, 7-3

processor-to-PCI-read buffer (PRPRB), 7-4

processor-to-PCI-write buffers (PRPWBs), 7-5

Burst operations

burst ordering

PCI cache wrap mode, 8-10

PCI linear incrementing, 8-10

PCI bus transfer, 8-7

Bus error handling registers, 5-26, 5-32

Bus error status registers, 13-5

Bus interface

bus ratios, 1-17

PCI bus arbitration unit, 1-14

peripheral logic bus, 1-10

peripheral logic bus interface, 2-9

INDEX

- Bus operations, PCI bus transactions, 8-12
- Bypass register, 15-22
- Byte
 - alignment, 8-11, B-2
 - byte enable signals, 3-8, 8-11, 8-28
 - ordering
 - big-endian mode, B-2
 - little-endian mode, B-6
 - most-significant byte/bit (MSB/msb), B-1
 - PCI bus, 8-2, B-1
 - processor core bus, B-1
 - PCI alignment, 8-11, B-2
- Byte ordering
 - mechanisms, B-1
- Byte-reverse instructions, A-22
- C**
 - $\overline{C}/\overline{BEn}$ (command/byte enable) signals, 3-8, 8-11, 8-28
 - Cache
 - cache coherency, 2-24
 - cache management instructions, A-25
 - central control unit (CCU), 2-24
 - overview, 2-8
 - processor core cache implementation, 2-20
 - Cache wrap mode, PCI, 8-10
 - \overline{CASn} (column address strobe) signals, 3-16
 - CDAR (current descriptor address register), 9-15
 - Central control unit (CCU)
 - cache coherency, 2-24
 - overview, 7-1
 - $\overline{CHKSTOP_IN}$ (checkstop in), 3-27
 - CKE (SDRAM clock enable) signal, 3-20
 - CKO (test clock) signal, 3-31
 - Clocks
 - clock subsystem block diagram, 3-32
 - clock synchronization, 3-34
 - clocking method, 3-32
 - clocking on the MPC8240, 3-32
 - DLL operation and locking, 3-33
 - examples, 3-34
 - signals, 3-30
 - Commands
 - PCI commands
 - $\overline{C}/\overline{BEn}$ signals, 3-8, 8-8
 - interrupt-acknowledge transaction, 8-24
 - PCI command register, 5-10, 8-19
 - special-cycle command, 8-25
 - Compare instructions, A-18
 - Completion, PCI transaction, 8-15
 - Configuration
 - configuration registers
 - 60x/PCI error address register, 5-33
 - accessing registers, 5-2–5-4
 - bus error status registers, 13-5
 - ECC single-bit error registers, 5-26, 13-6
 - emulation support, 5-33, 8-31
 - error detection registers, 5-29, 13-5
 - error enabling registers, 5-27
 - error handling registers, 5-26, 5-32
 - error status registers, 5-31, 8-28
 - memory bank enable register, 5-39
 - memory boundary registers, 5-35–??
 - memory control configuration registers, 5-41
 - memory interface configuration registers, 5-34
 - memory page mode register, 5-40
 - PCI command register, 5-10, 8-19
 - PCI status register, 5-11, 8-19
 - power management registers, 5-15–5-17
 - processor interface configuration registers, 5-22
 - processor/PCI error address register, 13-5
 - register access, 5-2–5-4
 - reserved bits, 5-1
 - summary of registers, list, 5-4, 5-6
 - configuration signals, 3-29
 - configuration space, 8-10
 - PCI addressing, 8-10
 - PCI configuration
 - configuration cycles
 - CONFIG_ADDR register, 5-1, 8-20
 - CONFIG_DATA register, 5-2, 8-21
 - configuration space header, 8-18
 - type 0 and 1 accesses, 8-20
 - configuration header summary, 5-8, 8-19
- Conventions, xxxvii
- \overline{CSn} (SDRAM command select) signals, 3-16
- D**
 - DA (debug address) signal, 3-28
 - DAR (destination address register), 9-17
 - Data bus
 - bus transaction errors, 13-6
 - shared data bus, 7-3
 - termination by \overline{TEA} , 13-9
 - Data high error capture monitor register, 15-20
 - Data high error injection mask register, 15-18
 - Data low error capture monitor register, 15-20
 - Debug
 - address, 15-4
 - address attribute signals, 15-1
 - address maps, 15-5
 - data high error capture monitor register, 15-20
 - data high error injection mask register, 15-18
 - data low error capture monitor register, 15-20
 - data path error injection/capture, 15-16
 - features list, 1-19
 - memory data path error capture monitor register, 15-19

INDEX

- memory debug address, 1-19
 - overview, 15-1
 - parity error capture monitor register, 15-20
 - parity error injection mask register, 15-19
 - PCI address attribute signals, 15-2
 - Decrementer interrupt, 14-2
 - Device drivers, posted writes, 7-5
 - DEVSEL (device select) signal, 3-9, 8-10
 - DHn/DLn (data bus) signals, 3-18
 - Disconnect, 7-4, 8-3, 8-15
 - DMA burst wrap, 6-22
 - DMA controller, 1-15
 - block diagram, 9-2
 - coherency, 9-5
 - DMA descriptors, 9-8
 - local memory to local memory transfers, 9-6
 - local memory to PCI, 9-6
 - modes
 - chaining mode, 9-4
 - direct mode, 9-4
 - operation, 9-3
 - overview, 9-1
 - PCI to local memory transfers, 9-6
 - PCI to PCI transfers, 9-6
 - register summary, 9-2
 - transfer types, 9-6
 - DMR (DMA mode register), 9-12
 - Doorbell registers
 - I₂O interface, 10-2
 - overview, 1-15
 - Doze mode, 1-18
 - DQMn (SDRAM data qualifier) signals, 3-17
 - DSR (DMA status register), 9-14
- ## E
- ECC single-bit error
 - registers, 5-26, 13-6
 - EDO DRAM interface
 - address multiplexing, 6-11
 - block diagram, 6-6
 - data interface, 6-14
 - DMA burst wrap, 6-22
 - ECC, 6-24
 - initialization, 6-16
 - organizations supported, 6-9
 - overview, 6-6
 - page mode retention, 6-23
 - parity, 6-23
 - power saving modes, 6-29
 - refresh timing, 6-28
 - RMW parity, 6-23
 - timing, 6-17
 - Effective address calculations, 2-18
 - Embedded utilities memory block, 4-16
 - Emulation mode
 - emulation support, 8-31
 - ESCR1/ESCR2 registers, 5-33
 - EPIC controller
 - block diagram, 12-3
 - EPIC vendor identification, 12-17
 - external interrupt source destination register, 12-24
 - external interrupt source vector register, 12-22
 - features list, 12-2
 - FPR, 12-15
 - global configuration register, 12-15
 - global timer base count, 12-20
 - global timer current count register, 12-20
 - global timer destination register, 12-22
 - global timer vector priority register, 12-21
 - internal block diagram, 12-8
 - internal interrupt destination register, 12-24
 - internal interrupt vector priority register, 12-24
 - interrupt protocol, 12-7
 - overview, 1-16, 12-1
 - pass-through capability, 12-10
 - processor current task priority register, 12-25
 - processor initialization, 12-18
 - processor interrupt acknowledge register, 12-25
 - programming interface, 12-13
 - registers, 12-15
 - registers not visible to the programmer
 - IPR, 12-9
 - IRR, 12-9
 - IS, 12-9
 - ISR, 12-9
 - serial interrupt configuration register, 12-16
 - serial interrupt interface, 12-10
 - serial interrupt source destination register, 12-24
 - signals, 12-2
 - spurious vector register, 12-18
 - timer frequency reporting register, 12-19
 - timers, 12-12
 - EPIC vendor identification register, 12-17
 - ErrDR1/ErrDR2 (error detection) registers, 5-29, 8-28, 13-5
 - ErrEnR1/ErrEnR2 (error enabling) registers, 5-27
 - Errors
 - EDO ECC, 6-24
 - error detection registers, 5-29, 8-28, 13-5
 - error enabling registers, 5-27
 - error handling
 - overview, 13-1
 - registers, 3-26, 5-26, 8-28, 13-3
 - error reporting, 13-6
 - address/data error, 8-16, 13-8
 - error detection registers, 5-29, 8-28, 13-5
 - errors within a nibble, 13-7
 - Flash write error, 13-6
 - master-abort transaction termination, 8-15, 13-9

INDEX

- nonmaskable interrupt, 3-26, 13-9
- PCI bus, 8-28, 13-4
- $\overline{\text{PERR}}$ and $\overline{\text{SERR}}$ signals, 8-29, 13-4
- system memory errors, 13-7
- unsupported bus transaction error, 13-6
- error status registers, 5-31, 8-28, 13-5
- FPM ECC, 6-24
- interrupt and error signals, 13-3
- overflow condition, 5-28, 13-8
- SDRAM ECC, 6-41
- ESCR1/ESCR2 (emulation support configuration) registers, 5-33
- Exceptions
 - bus errors, 3-26, 13-4
 - interrupt and error signals, 13-3
 - interrupt latencies, 13-10
 - interrupt priorities, 13-2
 - overview, 2-26
 - system reset interrupt, 13-3
- Exclusive access, PCI, 8-26
- Execution units, 2-6
- External control instructions, A-26
- External interrupt source destination register, 12-24
- External interrupt source vector priority register, 12-22

F

- Features lists
 - debug features, 1-19
 - EPIC controller, 12-2
 - I²C interface, 11-2
 - MPC8240 features, 1-3
 - peripheral logic, 1-11
 - processor core, 2-3
- Flash interface
 - address multiplexing, 6-65
 - operation, 6-61
 - overview, 6-61
 - timing, 6-66
 - write operations, 6-70
 - write timing, 6-71
- Floating-point model
 - floating-point unit overview, 2-7
 - FP arithmetic instructions, A-19
 - FP compare instructions, A-20
 - FP load instructions, A-23
 - FP move instructions, A-23
 - FP multiply-add instructions, A-20
 - FP rounding/conversion instructions, A-20
 - FP store instructions, A-23
 - FPSCR instructions, A-20
- $\overline{\text{FOE}}$ (flash output enable) signal, 3-22
- FPM interface
 - address multiplexing, 6-11

- block diagram, 6-6
- data interface, 6-14
- DMA burst wrap, 6-22
- ECC, 6-24
- initialization, 6-16
- organizations supported, 6-9
- overview, 6-6
- page mode retention, 6-23
- parity, 6-23
- power saving modes, 6-29
- refresh timing, 6-28
- RMW parity, 6-23
- timing, 6-17
- FPR (feature reporting register), 12-15
- FPSCR instructions, A-20
- $\overline{\text{FRAME}}$ signal, 3-10, 8-7
- Full-power mode, 1-18, 14-4

G

- G2 core, *see* Processor core
- Global configuration register, 12-15
- Global timer base count, 12-20
- Global timer current count register, 12-20
- Global timer destination register, 12-22
- Global timer vector priority register, 12-21
- $\overline{\text{GNT}}$ (PCI bus grant) signal, 3-10, 8-4

H

- Hard reset
 - configuration pins sampled, 3-35
 - $\overline{\text{HRST_CPU}}$ (hard reset (processor)), 3-25
 - $\overline{\text{HRST_CTRL}}$ (hard reset (peripheral logic)), 3-25
- HID0 (hardware implementation-dependent 0) registers, 2-13–2-17
 - doze bit, 14-4
 - DPM enable bit, 14-4
 - nap bit, 14-5
- HID1 (hardware implementation 1) register, 2-17
- HID2 (hardware implementation 2) register, 2-17

I

- I²C interface
 - arbitration procedure, 11-5
 - clock stretching, 11-7
 - clock synchronization, 11-6
 - data transfer, 11-5
 - features list, 11-2
 - handshaking, 11-6
 - I2CADR, 11-7
 - I2CCR, 11-10
 - I2CDR, 11-13
 - I2CFDR, 11-8

INDEX

- I²C SR, 11-12
 - operation, 11-3
 - overview, 1-16
 - programming model, 11-7
 - programming the I²C interface, 11-14
 - signals, 11-2
 - slave address transmission, 11-4
 - start condition, 11-4
 - stop condition, 11-5
 - system configuration, 11-2
- I²CADR (I²C address register), 11-7
- I²CCR (I²C control register), 11-10
- I²CDR (I²C data register), 11-13
- I²CFDR (I²C frequency divider register), 11-8
- I²CSR (I²C status register), 11-12
- I₂O interface
 - doorbell registers, 10-2
 - I₂O specification, 1-16
 - I₂O unit, 10-4
 - IFHPR, 10-14
 - IFQPR, 10-10
 - IFTPR, 10-15
 - IMIMR, 10-13
 - IMISR, 10-11
 - IPHPR, 10-15
 - IPTPR, 10-16
 - message registers, 10-1
 - MUCR, 10-19
 - OFHPR, 10-17
 - OFQPR, 10-11
 - OFTPR, 10-17
 - OMIMR, 10-9
 - OMISR, 10-8
 - OPHPR, 10-18
 - OPTPR, 10-18
 - outbound FIFOs, 10-7
 - overview, 10-1
 - PCI configuration identification, 10-4
 - QBAR, 10-20
 - registers
 - doorbell registers, 10-2
 - I₂O hardware registers, 10-5
 - message registers, 10-1
- IDSEL (ID select) signal, 3-15
- IEEE 1149.1 specifications
 - signals, 15-22
 - specification compliance, 15-22
- IFHPR (inbound free_FIFO head pointer register), 10-14
- IFQPR (inbound FIFO queue port register), 10-10
- IFTPR (inbound free_FIFO tail pointer register), 10-15
- ILR (PCI interrupt line register), 5-14
- IMIMR (inbound message interrupt mask register), 10-13
- IMISR (inbound message interrupt status register), 10-11
- Instructions
 - branch instructions, A-24
 - cache management instructions, A-25
 - condition register logical, A-24
 - external control, A-26
 - floating-point
 - arithmetic, A-19
 - compare, A-20
 - FP load instructions, A-23
 - FP move instructions, A-23
 - FP store instructions, A-23
 - FPSCR instructions, A-20
 - multiply-add, A-20
 - rounding and conversion, A-20
 - instruction timing overview, 2-33
 - instruction unit, 2-5
 - integer
 - arithmetic, A-17
 - compare, A-18
 - load, A-21
 - logical, A-18
 - multiple, A-22
 - rotate and shift, A-18–A-19
 - store, A-21
 - load and store
 - byte-reverse instructions, A-22
 - integer multiple instructions, A-22
 - string instructions, A-22
 - memory control, A-25
 - memory synchronization, A-22
 - PowerPC instruction set, 2-18
 - PowerPC instructions, list
 - form (format), A-27
 - function, A-17
 - legend, A-38
 - mnemonic, A-1
 - opcode, A-9
 - processor control, A-24
 - segment register manipulation, A-25
 - system linkage, A-24
 - TLB management instructions, A-25
 - trap instructions, A-24
- Integer arithmetic instructions, A-17
- Integer compare instructions, A-18
- Integer load instructions, A-21
- Integer logical instructions, A-18
- Integer multiple instructions, A-22
- Integer rotate and shift instructions, A-18–A-19
- Integer store instructions, A-21
- Integer unit, 2-6
- Interface
 - I₂O interface
 - doorbell registers, 10-2

INDEX

- I₂O unit, 10-4
- IFHPR, 10-14
- IFQPR, 10-10
- IFTP, 10-15
- IMIMR, 10-13
- IMISR, 10-11
- IPHPR, 10-15
- IPTPR, 10-16
- message registers, 10-1
- MUCR, 10-19
- OFHPR, 10-17
- OFQPR, 10-11
- OFTPR, 10-17
- OMIMR, 10-9
- OMISR, 10-8
- OPHPR, 10-18
- OPTPR, 10-18
- JTAG interface
 - block diagram, 15-22
 - boundary-scan registers, 15-23
 - bypass register, 15-22
 - instruction register, 15-23
 - registers, 15-22
 - signals, 15-22
 - status register, 15-23
 - TAP controller, 15-23
- memory interface
 - configuration registers, 5-34
 - ECC error, 13-7
 - error detection, 13-6
 - errors within a nibble, 13-7
 - features list, 1-12
 - Flash write error, 13-6
 - overview, 1-13
 - physical memory, 13-7
 - read data parity error, 13-7
 - refresh overflow error, 13-8
 - registers, 5-35, ??-5-41
 - select error, 13-7
 - signals, 3-16
 - system memory, 13-7
- PCI interface
 - address bus decoding, 8-9
 - address/data parity error, 8-16, 13-8
 - big-endian mode, B-3
 - burst operation, 8-7
 - bus arbitration, 8-4
 - bus arbitration unit, 1-14
 - bus commands, 8-8
 - bus error signals, 13-4
 - bus protocol, 8-7
 - bus transactions, 8-12, 8-24
 - byte alignment, 8-11, B-2
 - byte ordering, 8-2, B-1
 - $\overline{C}/\overline{BE}n$ signals, 8-28
 - cache wrap mode, 8-10
 - command encodings, 8-8
 - configuration cycles, 8-18
 - configuration header, 8-18
 - configuration space addressing, 8-10
 - data transfers, 8-7, 8-11
 - error detection and reporting, 8-28, 13-4, 13-8
 - error transactions, 8-28
 - exclusive access, 8-26
 - fast back-to-back transactions, 8-17
 - features list, 1-12
 - I/O space addressing, 8-10
 - linear incrementing, 8-10
 - master-abort transaction termination, 8-15, 13-9
 - memory space addressing, 8-10
 - MPC8240 as PCI bus master, 8-2
 - MPC8240 as PCI target, 8-3
 - nonmaskable interrupt, 3-26, 13-9
 - overview, 1-14, 8-1, 8-1
 - PCI Local Bus Specification*, 5-8
 - PCI special-cycle operations, 8-26
 - PCI-to-system memory read buffer (PCMRB), 7-7
 - PCI-to-system memory write buffers (PCMWBs), 7-8
 - processor-to-PCI read buffer (PRPRB), 7-4
 - processor-to-PCI-write buffers (PRPWBs), 7-5
 - read transactions, 8-12
 - registers, 5-8, 8-19, 13-8
 - retry PCI transactions, 8-15
 - signals, 3-7, 8-7-8-11
 - target-abort error, 8-16, 13-9
 - target-disconnect, 7-4, 8-3, 8-15
 - target-initiated termination, 8-16
 - transaction termination, 8-14
 - turnaround cycle, 8-11
 - write transactions, 8-13
- processor core
 - bus error handling registers, 5-26, 5-32
 - bus error signals, 13-3
 - bus error status register, 13-5
 - byte ordering, B-1
 - configuration registers, 5-22
 - error detection, 13-6
 - PCI buffers, 7-4
 - PICR1/PICR2 registers, 5-22
 - system memory buffer, 7-3
 - unsupported bus transactions error, 13-6
- Internal control
 - arbitration
 - in-order execution, 7-9
 - out-of-order execution, 7-1
 - buffers, 7-1

INDEX

Internal interrupt destination register, 12-24
Internal interrupt vector priority register, 12-24
Interrupt protocol, EPIC controller, 12-7
IPHPR (inbound post_FIFO head pointer register), 10-15
IPR (interrupt pending register), 12-9
IPTPR (inbound post_FIFO tail pointer register), 10-16
 $\overline{\text{IRDY}}$ (initiator ready) signal, 3-12, 8-7
IRQ n (discrete interrupt), 3-22
IRR (interrupt request register) register, 12-9
IS (interrupt selector) register, 12-9
ISR (in-service register), 12-9
ITWR (inbound translation window register), 4-14

J

JTAG interface
 block diagram of JTAG interface, 15-22
 boundary-scan registers, 15-23
 bypass register, 15-22
 instruction register, 15-23
 JTAG registers, 15-22
 JTAG signals, 15-22
 status register, 15-23
 TAP controller, 15-23

L

$\overline{\text{L_INT}}$ (local interrupt) signal, 3-23
Little-endian mode
 accessing configuration registers, 5-2
 aligned scalars, address modification, B-6
 byte lane translation, B-7
 byte ordering, B-6
 DMA descriptors, 9-11
 LE_MODE bit, 5-23, B-6
 PCI bus, 8-2, B-1
 PCI I/O space, B-12
 PCI memory space, B-9
LMBAR (local memory base address register), 4-13, 5-14
Load/store
 byte-reverse instructions, A-22
 floating-point load instructions, A-23
 floating-point move instructions, A-23
 floating-point store instructions, A-23
 integer load instructions, A-21
 integer store instructions, A-21
 load/store multiple instructions, A-22
 memory synchronization instructions, A-22
 string instructions, A-22
 $\overline{\text{LOCK}}$ signal, 3-12, 8-26

M

MAA (memory address attribute) signals, 3-27, 15-1
Master-abort, PCI, 8-15, 13-9
MCCRN (memory control configuration) registers, 5-41–5-50
 $\overline{\text{MCP}}$ (machine check), 3-25
Memory data path error capture monitor register, 15-19
Memory interface
 address signals, 6-5
 block diagram, 6-3
 configuration at reset, 6-6
 configuration registers, 5-34
 DMA burst wrap, 6-22
 ECC error, 13-7
 EDO DRAM interface, 6-6
 error detection, 13-6
 errors within a nibble, 13-7
 features list, 1-12
 Flash interface, 6-61
 Flash write error, 13-6
 FPM interface, 6-6
 memory attribute signals, 1-19
 overview, 1-13, 6-1
 physical memory, 13-7
 Port X, 6-72
 read data parity error, 13-7
 refresh overflow error, 13-8
 registers
 memory bank enable register, 5-39
 memory boundary registers, 5-35–??
 memory control configuration registers, 5-41
 memory page mode register, 5-40
 ROM interface, 6-61
 SDRAM interface, 6-31
 select error, 13-7
 signal summary, 6-4
 signals, 3-16
 system memory, 13-7
Memory management unit (MMU)
 overview, 2-8, 2-31
Memory page mode register, 5-40
Memory synchronization
 instructions, A-22
Message unit
 overview, 1-15
Message unit, *see* I₂O interface, 10-1
 $\overline{\text{MIV}}$ (memory interface valid), 3-28
 $\overline{\text{MIV}}$ (memory interface valid) signal, 1-20, 15-7
Modes
 doze mode, 1-18
 full-power mode, 1-18
 munged little-endian byte ordering, B-1
 nap mode, 1-18

INDEX

- sleep mode, 1-18
- MPC8240
 - aligned scalars, address modification, B-6
 - differences with the processor core, 2-34
 - implementation-specific registers, 2-13
 - MMU features, 2-32
 - PCI bus master, 8-2
 - PCI target, 8-3
 - peripheral logic block diagram, 1-11
 - possible applications, 1-5
 - processor core block diagram, 2-2
 - processor core differences, 2-34
 - uses for the MPC8240, 1-5
- MUCR (messaging unit control register), 10-19
- Munged little endian mode, *see* PowerPC little-endian (PPC-LE) mode
- Munged memory image, LE mode, B-7
- Munging, definition, B-1

N

- Nap mode, 1-18
 - description, 1-17
 - special cycle, PCI, 8-25
- NDAR (next descriptor address register), 9-18
- NMI (nonmaskable interrupt) signal, 3-26, 13-5, 13-9

O

- OFHPR (outbound free_FIFO head pointer register), 10-17
- OFQPR (outbound FIFO queue port register), 10-11
- OFTPR (outbound free_FIFO tail pointer register), 10-17
- OMBAR (outbound memory base address register), 4-15
- OMIMR (outbound message interrupt mask register), 10-9
- OMISR (outbound message interrupt status register), 10-8
- OPHPR (outbound post_FIFO head pointer register), 10-18
- Optional instructions, A-38
- OPTPR (outbound post_FIFO tail pointer register), 10-18
- OSC_IN (system clock input) signal, 3-30
- OTWR (outbound translation window register), 4-15

P

- PAR (PCI parity) signal, 3-13, 8-28
- Parity error capture monitor register, 15-20
- Parity error injection mask register, 15-19
- PAR n (data parity/ECC) signals, 3-19
- PBCCR (PCI base class code register), 5-13

- PCI configuration
 - PCI addressing, 8-10
- PCI interface
 - address bus decoding, 8-9
 - address/data parity error, 8-16, 13-8
 - big-endian mode, four-byte transfer, B-3
 - burst operation, 8-7
 - bus arbitration, 8-4
 - bus arbitration unit, 1-14
 - bus commands, 8-8
 - bus error signals, 13-4
 - bus protocol, 8-7
 - bus transactions
 - fast back-to-back transactions, 8-17
 - interrupt-acknowledge transaction, 8-24
 - legend for timing diagrams, 8-12
 - read transactions, 8-12
 - special-cycle transaction, 8-25
 - transaction termination, 8-14
 - write transactions, 8-13
 - byte alignment, 8-11, B-2
 - byte ordering, 8-2, B-1
 - cache wrap mode, 8-10
 - configuration cycles, 8-18
 - configuration header, 8-18
 - configuration space, 8-10
 - data transfers, 8-7
 - error detection and reporting, 8-28, 13-4, 13-8
 - error transactions, 8-28
 - exclusive access, 8-26
 - features list, 1-12
 - I/O space addressing, 8-10
 - linear incrementing, 8-10
 - little-endian mode transfers to I/O space, B-12
 - little-endian mode transfers to memory space, B-9
 - master-abort transaction termination, 8-15, 13-9
 - memory space addressing, 8-10
 - MPC8240 as PCI bus master, 8-2
 - MPC8420 as PCI target, 8-3
 - nonmaskable interrupt, 3-26, 13-9
 - overview, 1-14, 8-1
 - PCI attribute signals, 1-19
 - PCI command encodings, 8-8
 - PCI commands
 - interrupt-acknowledge transaction, 8-24
 - special-cycle command, 8-25
 - PCI Local Bus Specification*, 5-8
 - PCI special-cycle operations, 8-26
 - PCI-to-ISA bridge, 13-5
 - processor-to-PCI read buffer (PRPRB), 7-4
 - processor-to-PCI-write buffers (PRPWBs), 7-5
 - registers
 - bus error status register, 5-32, 13-8
 - CONFIG_ADDR register, 8-20

INDEX

- CONFIG_DATA register, 5-2, 8-21
 - configuration header summary, 5-8, 8-19
 - PCI commands register, 5-10, 8-19
 - status register, 5-11, 8-19
 - retry PCI transactions, 8-15
 - signals
 - DEVSEL, 3-9, 8-10
 - error reporting signals, 13-4
 - FRAME, 3-10, 8-7
 - GNT, 3-10, 8-4
 - IRDY, 3-12, 8-7
 - LOCK, 3-12, 8-26
 - PERR, 3-13, 8-29, 13-4
 - REQ, 3-11, 8-4
 - SERR, 3-13, 8-29, 13-4
 - TRDY, 3-14, 8-7
 - target-abort error, 8-16, 13-9
 - target-disconnect, 7-4, 8-3, 8-15
 - target-initiated termination, 8-16
 - turnaround cycle, 8-11
 - PCI interface
 - registers
 - CONFIG_ADDR register, 5-1
 - PCI_CLK (PCI clock), 3-30
 - PCI_SYNC_IN (PCI feedback clock), 3-31
 - PCI_SYNC_OUT (PCI clock synchronize out), 3-31
 - PCLSR (PCI cache line size register), 5-13
 - PCSRBAR (PCSR base address register), 5-14
 - Peripheral logic
 - block diagram, 1-11
 - bus interface, 2-9
 - bus operation, 1-10
 - control and status registers, 4-18
 - DMA controller, 1-15
 - features list, 1-11
 - major functional units, 1-12
 - overview, 1-10
 - power management modes, 1-18
 - PERR (PCI parity error) signal, 3-13, 8-29, 13-4
 - Phase locked loop, 14-5
 - PICRs (processor interface configuration registers)
 - PICR1 register
 - bit settings/overview, 5-22
 - CF_BREAD_WS bit, 5-22
 - FLASH_WR_EN bit, 5-23, 13-6
 - LE_MODE (endian mode) bit, 5-23
 - MCP_EN bit, 3-26, 13-4
 - speculative PCI reads bit, 5-24
 - ST_GATH_EN bit, 5-23
 - PICR2 register
 - bit settings/overview, 5-24
 - CF_APARK bit, 5-24
 - CF_APHASE_WS bit, 5-26
 - CF_SNOOP_WS bit, 5-25
 - FLASH_WR_LOCKOUT bit, 5-25, 13-6
 - PIR (programming interface registers), 5-12
 - PLL_CFG (PLL configuration) signals, 3-29
 - PLTR (PCI latency timer register), 5-13
 - PMAA (PCI memory address attribute) signals, 3-28
 - Port X interface
 - block diagram, 6-72
 - overview, 6-72
 - Power management
 - doze mode, 14-4
 - doze, nap, sleep, DPM bits, 2-13
 - dynamic power management, 14-2
 - full-power mode, 14-4
 - nap mode, 8-26, 14-5
 - overview, 1-17, 14-1
 - PCI special-cycle operations, 8-25
 - peripheral logic programmable power modes, 1-18
 - PMCR registers
 - overview, 5-15-??
 - PMCR1
 - bit settings, 5-16
 - PM bit, 5-16
 - processor core, 14-1
 - programmable power modes, 1-18, 14-2, 14-8
 - sleep mode, 5-16, 14-5
 - software considerations, 14-6
- Power-on reset
- configuration pins sampled, 3-35
 - initialization at power-on reset (POR), 13-3
 - memory interface configuration, 6-6
 - output signal state, 3-7
 - SDRAM initialization, 6-46
- PowerPC architecture
- instruction list, A-1, A-9, A-17
 - instruction set, 2-18
- Processor control instructions, A-24
- Processor core
- block diagram, 2-2
 - bus error signals, 13-3
 - bus error status register, 13-5
 - byte ordering, B-1
 - cache implementation, 2-20
 - cache units, 2-8
 - configuration registers, 5-22
 - control and status registers, 4-17, 4-17
 - differences with the MPC8240, 2-34
 - dispatch unit, 2-5
 - error detection, 13-6
 - execution units, 2-6
 - features list, 2-3
 - floating-point unit, 2-7
 - instruction queue, 2-5
 - instruction timing, 2-33
 - instruction unit, 2-5
 - integer unit, 2-6
 - memory management unit, 2-8, 2-31

INDEX

- overview, 1-7, 2-1
- PCI buffers, 7-4, 7-4
- programmable parameters
 - parking, 5-22
 - PICR1/PICR2 registers, 5-22
- programming model, 2-10
- shared data bus, 7-3
- system memory buffer, 7-3
- unsupported bus transactions error, 13-6
- Processor current task priority register, 12-25
- Processor initialization register, 12-18
- Processor interrupt acknowledge register, 12-25
- Programmable power states
 - doze mode, 14-4
 - full-power mode (DPM enabled/disabled), 14-4
 - nap mode, 14-5
 - sleep mode, 14-5
- PVR (processor version register), 2-18

Q

- \overline{QACK} (quiesce acknowledge) signal, 3-27
- QBAR (queue base address register), 10-20

R

- RAM access time, 5-44
- \overline{RASn} (row address strobe) signals, 3-16
- \overline{RCSn} (ROM bank select) signals, 3-21
- Registers
 - 60x/PCI error address register, 5-33
 - address translation registers, 4-12
 - BCR, 9-18
 - bus error status registers, 13-5
 - CDAR, 9-15
 - clearing bits, 5-11
 - configuration registers, 5-1
 - DAR, 9-17
 - data high error capture monitor, 15-20
 - data high error injection mask register, 15-18
 - data low error capture monitor, 15-20
 - DCMP, 2-13
 - DMA controller, 9-2
 - DMISS, 2-13
 - DMR, 9-12
 - doorbell registers, 1-15, 10-2
 - DSR, 9-14
 - ECC single-bit error registers, 5-26, 13-6
 - EPIC vendor identification, 12-17
 - error detection registers, 5-29, 13-5
 - error enabling registers, 5-27
 - error handling registers, 3-26, 5-26, 5-32, 8-28, 13-3
 - error status registers, 5-31, 8-28

- ESCRs (emulation support configuration)
 - registers, 5-33
- external interrupt source destination, 12-24
- external interrupt source vector priority, 12-22
- FPR, 12-15
- global configuration register, 12-15
- global timer base count, 12-20
- global timer current count, 12-20
- global timer destination, 12-22
- global timer vector priority, 12-21
- HASH1, 2-13
- HASH2, 2-13
- HID0, 2-13
- HID1, 2-17
- HID2, 2-17
- I2CADR, 11-7
- I2CCR, 11-10
- I2CDR, 11-13
- I2CFDR, 11-8
- I2CSR, 11-12
- I₂O hardware registers, 10-5
- IABR, 2-13
- ICMP, 2-13
- IFHPR, 10-14
- IFQPR, 10-10
- IFTPR, 10-15
- ILR, 5-14
- IMIMR, 10-13
- IMISR, 10-11
- IMISS, 2-13
- implementation-specific registers, 2-13
- internal interrupt destination, 12-24
- internal interrupt vector priority, 12-24
- IPHPR, 10-15
- IPR, 12-9
- IPTPR, 10-16
- IRR, 12-9
- IS, 12-9
- ISR, 12-9
- ITWR, 4-14
- JTAG
 - boundary-scan registers, 15-23
 - bypass register, 15-22
 - instruction register, 15-23
 - status register, 15-23
- LMBAR, 4-13, 5-14
- MCCR_n (memory control configuration)
 - registers, 5-41–5-50
- memory bank enable register, 5-39
- memory boundary registers, 5-35
- memory data path error capture monitor, 15-19
- memory page mode register, 5-40
- message registers, 10-1
- MICR_n (memory interface configuration)

INDEX

- registers, 5-34
- MUCR, 10-19
- NDAR, 9-18
- non-programmable registers
 - IPR, 12-9
 - IRR, 12-9
 - IS, 12-9
 - ISR, 12-9
- OFHPR, 10-17
- OFQPR, 10-11
- OFTPR, 10-17
- OMBAR, 4-15
- OMMMR, 10-9
- OMISR, 10-8
- OPHPR, 10-18
- optional register, BIST control, 5-9
- OPTPR, 10-18
- OTWR, 4-15
- parity error capture monitor, 15-20
- parity error injection mask register, 15-19
- PBCCR, 5-13
- PCI access to configuration registers, 5-2
- PCI arbiter control register, 5-15
- PCI command register, 5-10, 8-19
- PCI registers, 5-8
- PCI status register, 5-11, 8-19
- PCLSR, 5-13
- PCSRBAR, 5-14
- peripheral control and status registers, 4-18
- PICR*n* (processor interface configuration) registers, 5-22
- PIR, 5-12
- PLTR, 5-13
- power management registers, 5-15–5-17
- processor access to configuration registers, 5-1
- processor core control and status registers, 4-17, 4-17
- processor current task priority, 12-25
- processor initialization, 12-18
- processor interrupt acknowledge, 12-25
- processor/PCI error address register, 13-5
- PVR, 2-18
- QBAR, 10-20
- RPA, 2-13
- SAR, 9-16
- serial interrupt configuration register, 12-16
- serial interrupt source destination, 12-24
- setting bits, 5-11
- spurious vector, 12-18
- timer frequency reporting, 12-19
- REQ (PCI bus request) signal, 3-11, 8-4
- Reservation set, lwarx/stwcx., 2-24
- Reset, memory interface configuration, 6-6
- ROM interface
 - address multiplexing, 6-65

- block diagram, 6-61
- operation, 6-61
- overview, 6-61
- timing, 6-66
- write operations, 6-70
- write timing, 6-71

Rotate and shift instructions, A-18–A-19

S

- S_CLK (serial interrupt clock) signal, 3-23
- S_FRAME (serial interrupt frame) signal, 3-23
- S_INT (serial interrupt stream) signal, 3-23
- S_RST (serial interrupt reset) signal, 3-23
- SAR (source address register), 9-16
- SCL (serial clock) signal, 3-24
- SDA (serial data) signal, 3-24
- SDCAS (SDRAM column address strobe) signal, 3-21
- SDRAM interface
 - address multiplexing, 6-36
 - block diagram, 6-31
 - data interface, 6-39
 - ECC, 6-41
 - initialization, 6-46
 - JEDEC functionality on the MPC8240, 6-47
 - operation, 6-31
 - organizations supported, 6-35
 - overview, 6-31
 - page mode retention, 6-44
 - parity, 6-53
 - power saving modes, 6-58
 - refresh, 6-56
 - registered DIMM mode, 6-55
 - RMW parity, 6-53
 - system configuration, 6-40
 - timing, 6-49
- SDRAM_CLK (SDRAM clock outputs), 3-31
- SDRAM_SYNC_IN (SDRAM feedback clock), 3-31
- SDRAM_SYNC_OUT (SDRAM clock synchronize out), 3-31
- SDRAS (SDRAM row address strobe) signal, 3-21
- Segment registers
 - SR manipulation instructions, A-25
 - T bit, 3
- Serial interrupt configuration register, 12-16
- Serial interrupt source destination register, 12-24
- SERR (system error) signal, 3-13, 8-29, 13-4
- Signals
 - AD*n*, 3-8, 8-10
 - alternate functions list, 3-4
 - AR*n*, 3-20
 - AS, 3-22
 - C/BEN, 3-8, 8-11, 8-28
 - CAS*n*, 3-16
 - CHKSTOP_IN, 3-27

INDEX

- CKE, 3-20
 - CKO, 3-31
 - clock signals, 3-30
 - configuration pins sampled at reset, 3-35
 - configuration signals, 3-29
 - cross-reference list, 3-4
 - \overline{CS}_n , 3-16
 - DA, 3-28
 - debug address attribute signals, 15-1
 - debug signals, 3-27
 - \overline{DEVSEL} , 3-9, 8-10
 - DHn/DLn, 3-18
 - DQMn, 3-17
 - EPIC control, 3-22
 - EPIC controller, 12-2
 - error signals, 13-3
 - FOE, 3-22
 - FRAME, 3-10, 8-7
 - GNT, 3-10, 8-4
 - HRST_CPU, 3-25
 - HRST_CTRL, 3-25
 - I2C control signals, 3-23
 - I²C interface, 11-2
 - IDSEL, 3-15
 - IEEE 1149.1 interface, 15-22
 - interrupt and error signals, 13-3
 - interrupt signals, 13-3
 - IRDY, 3-12, 8-7
 - IRQn, 3-22
 - JTAG signals, 15-22
 - $\overline{L_INT}$, 3-23
 - LOCK, 3-12, 8-26
 - MAA, 3-27, 15-1
 - MCP, 3-25
 - memory attribute signals, 1-19
 - memory interface, 3-16
 - memory interface address signals, 6-5
 - memory interface signals, 6-4
 - MIV, 1-20, 3-28, 15-7
 - NMI, 3-26, 13-5, 13-9
 - OSC_IN, 3-30
 - output signal states at power-on reset, 3-7
 - PAR (PCI parity), 3-13, 8-28
 - PARn (data parity/ECC), 3-19
 - PCI address attribute signals, 15-2
 - PCI attribute signals, 1-19
 - PCI interface signals, 3-7
 - PCI_CLK, 3-30
 - PCI_SYNC_IN, 3-31
 - PCI_SYNC_OUT, 3-31
 - \overline{PERR} , 3-13, 8-29, 13-4
 - PLL_CFG, 3-29
 - PMMA, 3-28
 - power management signals, 3-24
 - \overline{QACK} , 3-27
 - RASn, 3-16
 - \overline{RCS}_n (ROM bank select), 3-21
 - REQ, 3-11, 8-4
 - S_CLK, 3-23
 - $\overline{S_FRAME}$, 3-23
 - S_INT, 3-23
 - S_RST, 3-23
 - SCL, 3-24
 - SDA, 3-24
 - \overline{SDCAS} , 3-21
 - SDRAM_CLK, 3-31
 - SDRAM_SYNC_IN, 3-31
 - SDRAM_SYNC_OUT, 3-31
 - \overline{SDRAS} , 3-21
 - serial interrupt mode signals, 3-23
 - SERR, 3-13, 8-29, 13-4
 - signal groupings, 3-3
 - \overline{SMI} , 3-26
 - \overline{SRESET} , 3-25
 - \overline{STOP} , 3-14, 8-11
 - system control signals, 3-24
 - TBEN, 3-27
 - TCK (JTAG test clock), 3-29, 15-22
 - TDI (JTAG test data input), 3-29, 15-22
 - TDO (JTAG test data output), 3-30, 15-22
 - TMS (JTAG test mode select), 3-30, 15-22
 - \overline{TRDY} , 3-14, 8-7
 - \overline{TRST} (JTAG test reset), 3-30, 15-22
 - WE, 3-17
 - Sleep mode, 1-18
 - description, 1-17
 - PMCR bit settings, 5-16
 - \overline{SMI} (system management interrupt), 3-26
 - Snooping
 - snoop response, 7-9
 - Spurious vector register, 12-18
 - \overline{SRESET} (soft reset), 3-25
 - Status register, PCI, 5-11, 8-19
 - \overline{STOP} signal, 3-14, 8-11
 - String instructions, A-22
 - Synchronization
 - memory synchronization instructions, A-22
 - System linkage instructions, A-24
 - System management interrupt, 14-2
 - System reset
 - system reset interrupt, 13-3
- ## T
- Target-abort error, 8-16, 13-9
 - Target-disconnect, *see* PCI interface
 - Target-initiated termination
 - description, 7-4, 8-3, 8-15
 - PCI status register, 8-16

INDEX

TBEN (time base enable) signal, 3-27
TCK (JTAG test clock) signal, 3-29, 15-22
TDI (JTAG test data input) signal, 3-29, 15-22
TDO (JTAG test data output) signal, 3-30, 15-22
Termination
 completion, PCI transaction, 8-15
 master-abort, PCI, 8-15, 13-9
 target-disconnect, PCI, 7-4, 8-3, 8-15
 target-initiated termination, 8-16
 termination by TEA, 13-9
 termination of PCI transaction, 8-14
 timeout, PCI transaction, 8-15
Timeout, PCI transaction, 8-15
Timer frequency reporting register, 12-19
TLB
 invalidate, A-25
 TLB management instructions, A-25
TMS (JTAG test mode select) signal, 3-30, 15-22
Transactions
 fast back-to-back transactions, PCI bus, 8-17
 PCI bus transactions, 8-12
 PCI transaction termination, 8-14
 $\overline{\text{TRDY}}$ (target ready) signal, 3-14, 8-7, 8-7, 8-14
 $\overline{\text{TRST}}$ (JTAG test reset) signal, 3-30, 15-22
Turnaround cycle and PCI bus, 8-11

W

$\overline{\text{WE}}$ (write enable) signal, 3-17

INDEX

Overview	1
PowerPC Processor Core	2
Signal Descriptions and Clocking	3
Address Maps	4
Configuration Registers	5
MPC8240 Memory Interface	6
Central Control Unit	7
PCI Bus Interface	8
DMA Controller	9
Message Unit (I ₂ O)	10
I ² C Interface	11
Embedded Programmable Interrupt Controller (EPIC)	12
Error Handling and Exceptions	13
Power Management	14
Debug Features	15
PowerPC Instruction Set Listings	A
Bit and Byte Ordering	B
Glossary of Terms and Abbreviations	GLO
Index	IND

- 1** Overview
- 2** PowerPC Processor Core
- 3** Signal Descriptions and Clocking
- 4** Address Maps
- 5** Configuration Registers
- 6** MPC8240 Memory Interface
- 7** Central Control Unit
- 8** PCI Bus Interface
- 9** DMA Controller
- 10** Message Unit (I₂O)
- 11** I²C Interface
- 12** Embedded Programmable Interrupt Controller (EPIC)
- 13** Error Handling and Exceptions
- 14** Power Management
- 15** Debug Features

A PowerPC Instruction Set Listings

B Bit and Byte Ordering

GLO Glossary of Terms and Abbreviations

IND Index

Attention!

This book is a companion to the *PowerPC Microprocessor Family: The Programming Environments*, referred to as *The Programming Environments Manual*. Note that the companion *Programming Environments Manual* exists in two versions. See the Preface for a description of the following two versions:

- *PowerPC Microprocessor Family: The Programming Environments*, Rev 1
Order #: MPCFPE/AD
- *PowerPC Microprocessor Family: The Programming Environments for 32-Bit Microprocessors*, Rev 1
Order #: MPCFPE32B/AD

Call the Motorola LDC at 1-800-441-2447 or contact your local sales office to obtain copies.



Motorola Literature Distribution Centers:

USA/EUROPE: Motorola Literature Distribution;

P.O. Box 5405, Denver, Colorado 80217

Tel.: 1-800-441-2447 or 1-303-675-2140;

World Wide Web Address: <http://ldc.nmd.com/>

JAPAN: Nippon Motorola Ltd SPD, Strategic Planning Office

4-32-1, Nishi-Gotanda Shinagawa-ku, Tokyo 141, Japan Tel.: 81-3-5487-8488

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Centre 2,

Dai King Street, Tai Po Industrial Estate

Tai Po, New Territories, Hong Kong

Mfax™: RMFAX0@email.sps.mot.com;

TOUCHTONE 1-602-244-6609;

US & Canada ONLY (800) 774-1848;

World Wide Web Address: <http://sps.motorola.com/mfax>

INTERNET: <http://www.motorola.com/sps>

Technical Information:

Motorola Inc. SPS Customer Support Center 1-800-521-6274;

electronic mail address: crc@wmkmail.sps.mot.com.

Document Comments: FAX (512) 895-2638,

Attn: RISC Applications Engineering.

World Wide Web Addresses:

<http://www.motorola.com/PowerPC/>

<http://www.motorola.com/netcomm/>

<http://www.motorola.com/ColdFire/>

MPC8240UM/D

PowerPC™

1ATX45554-0 PRINTED IN USA 8/99 IMPERIAL LITHO 43218 3,000 LITRISC

