DECchip 21066 and DECchip 21068 Microprocessors

Hardware Reference Manual

Order Number: To be supplied

This is an advance preliminary version of the hardware reference manual.

First Draft November 5, 1993

Preliminary

Digital Equipment Corporation
Maynard, Massachusetts

First Printing, 1993

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

While Digital believes the information included in this publication is correct as of the date of publication, it is subject to change without notice.

Digital Equipment Corporation makes no representations that the use of its products in the manner described in this publication will not infringe on existing or future patent rights, nor do the descriptions contained in this publication imply the granting of licenses to make, use, or sell equipment or software in accordance with the description.

© Digital Equipment Corporation 1993.

All Rights Reserved.
Printed in U.S.A.

Alpha AXP, AXP, the AXP logo, DEC, DECchip, DECsystem, Digital, the DIGITAL logo, TurboChannel, VAX, and OpenVMS are trademarks of Digital Equipment Corporation.

Intel is a trademark of Intel Corporation.
Microsoft is registered trademark of Microsoft Corporation
OSF/1 is a registered trademark of Open Software Foundation, Inc.

This document was prepared using VAX DOCUMENT, Version 2.1.

Preliminary

Contents

Preface xvii

Conventions xix

1 Introduction

1.1 Architecture 1–1
1.2 Chip Features 1–2

2 Internal Architecture

2.1 Overview 2–1
2.2 Instruction Fetch and Decode Unit 2–3
2.2.1 Instruction Fetch and Decode 2–3
2.2.2 Branch Prediction Logic 2–3
2.2.3 Instruction Translation Buffers 2–4
2.2.4 Interrupt Logic 2–5
2.2.5 Performance Counters 2–7
2.3 Integer Execution Unit 2–8
2.4 Load and Store Unit 2–8
2.4.1 Data Translation Buffer (DTB) 2–9
2.4.2 Load Silos 2–10
2.4.3 Write Buffer 2–11
2.4.3.1 Memory Barrier Instruction 2–12
2.5 Floating-Point Unit 2–12
2.5.1 IEEE Floating-Point Conformance 2–13
2.6 Cache Organization 2–15
2.6.1 Data Cache 2–15
2.6.2 Instruction Cache 2–15
2.7 Memory Controller 2–15
2.8 PCI I/O Controller 2–16
2.9 Pipeline Organization 2–16

Preliminary

2.9.1 Static and Dynamic Stages 2–17
2.9.2 Aborts 2–18
2.9.3 Nonissue Conditions 2–19
2.10 Scheduling and Issuing Rules 2–19
2.10.1 Instruction Class Definition 2–19
2.10.2 Producer-Consumer Latency 2–20
2.10.3 Producer-Producer Latency 2–22
2.10.4 Instruction Issue Rules 2–22
2.10.5 Dual-Issue Table 2–23
2.11 PALcode 2–24
2.11.1 Required PALcode Instructions 2–25
2.11.2 Architecturally Reserved PALcode Instructions 2–25
2.12 Physical Address Space 2–26
2.13 Lock Registers 2–27

3 Privileged Architecture Library Code

3.1 Introduction 3–1
3.2 PALcode 3–1
3.3 PALmode Environment 3–2
3.4 Invoking PALcode 3–3
3.4.1 CALL_PAL Instruction 3–4
3.5 PALcode Entry Points 3–5
3.6 PALmode Restrictions 3–8
3.6.1 Translation Buffer Miss Flows 3–15
3.6.1.1 Instruction Translation Buffer Miss 3–15
3.6.1.2 Data Translation Buffer Miss 3–16
3.7 Implementation of the Architecturally Reserved Instructions . . . 3–17
3.7.1 HW_MFPR and HW_MTPR Instructions 3–18
3.7.2 HW_LD and HW_ST Instructions 3–20
3.7.3 HW_REI Instruction 3–21

4 Internal Processor Registers

4.1 Introduction 4–1
4.2 Ibox Internal Processor Registers 4–1
4.2.1 Translation Buffer Tag Register (TB_TAG) 4–1
4.2.2 Instruction Translation Buffer Page Table Entry Register (ITB_PTE) 4–2
4.2.3 Instruction Cache Control and Status Register (ICCSR) 4–4
4.2.3.1 Performance Counters 4–7
4.2.4 Instruction Translation Buffer Page Table Entry Temporary Register (ITB_PTE_TEMP) 4–9

Preliminary

4.2.5 Exceptions Address Register (EXC_ADDR) 4–10
4.2.6 Clear Serial Line Interrupt Register (SL_CLR) 4–11
4.2.7 Serial Line Receive Register (SL_RCV) 4–12
4.2.8 Instruction Translation Buffer ZAP Register (ITBZAP) 4–13
4.2.9 Instruction Translation Buffer ASM Register (ITBASM) 4–13
4.2.10 Instruction Translation Buffer IS Register (ITBIS) 4–13
4.2.11 Processor Status Register (PS) 4–13
4.2.12 Exception Summary Register (EXC_SUM) 4–14
4.2.13 PAL Base Address Register (PAL_BASE) 4–16
4.2.14 Hardware Interrupt Request Register (HIRR) 4–16
4.2.15 Software Interrupt Request Register (SIRR) 4–18
4.2.16 Asynchronous Trap Request Register (ASTRR) 4–19
4.2.17 Hardware Interrupt Enable Register (HIER) 4–20
4.2.18 Software Interrupt Enable Register (SIER) 4–22
4.2.19 AST Interrupt Enable Register (ASTER) 4–23
4.2.20 Serial Line Transmit Register (SL_XMIT) 4–24
4.3 Abox Internal Processor Registers 4–25
4.3.1 Translation Buffer Control Register (TB_CTL) 4–25
4.3.2 Data Translation Buffer Page Table Entry Register (DTB_PTE) 4–26
4.3.3 Data Translation Buffer Page Table Entry Temporary Register (DTB_PTE_temp) 4–26
4.3.4 Memory Management Control and Status Register (MM_CSR) 4–27
4.3.5 Virtual Address Register (VA) 4–28
4.3.6 Data Translation Buffer ZAP Register (DTBZAP) 4–28
4.3.7 Data Translation Buffer ASM Register (DTBASM) 4–29
4.3.8 Data Translation Buffer Invalidate Single Register (DTBIS) 4–29
4.3.9 Flush Instruction Cache Register (FLUSH_IC) 4–29
4.3.10 Flush Instruction Cache ASM Register (FLUSH_IC_ASM) 4–29
4.3.11 Abox Control Register (ABOX_CTL) 4–29
4.3.12 Alternate Processor Mode Register (ALT_MODE) 4–31
4.3.13 Cycle Counter Register (CC) 4–32
4.3.14 Cycle Counter Control Register (CC_CTL) 4–33
4.4 PAL_TEMP Internal Processor Registers 4–34
4.4.1 Data Cache Status Register (DC_STAT) 4–34
4.5 Internal Processor Registers Reset State 4–35

Preliminary

5 Memory Controller

5.1 Introduction 5–1
5.1.1 Supported Memory Types 5–2
5.1.2 Supported Memory Sizes 5–3
5.1.3 Backup Cache 5–3
5.1.3.1 Backup Cache Initialization 5–4
5.1.4 Programmable Configuration 5–4
5.1.5 Error Correction and Detection 5–4
5.1.6 Graphics Features 5–6
5.2 Memory Controller Operation 5–6
5.2.1 Masked Writes 5–6
5.2.2 Page Mode 5–7
5.2.3 Register Access 5–7
5.3 Memory Error Conditions 5–9
5.3.1 Correctable Read Error 5–9
5.3.2 Uncorrectable Read Error 5–9
5.3.3 Uncorrectable Write Error 5–10
5.3.4 Bcache Tag Parity Error 5–10
5.3.5 Nonexistent Memory Address Error 5–10
5.4 Graphics Operations 5–11
5.4.1 Simple Frame Buffer Mode 5–11
5.4.2 Transparent Stipple Mode 5–11
5.5 Registers 5–12
5.5.1 Bank Configuration Registers (BCR3–BCR0) 5–13
5.5.2 Bank Address Mask Registers (BMR0-BMR3) 5–16
5.5.3 Bank Timing Registers (BTR0-BTR3) 5–18
5.5.4 Global Timing Register (GTR) 5–20
5.5.5 Error Status Register (ESR) 5–21
5.5.6 Error Address Register (EAR) 5–24
5.5.7 Cache Register (CAR) 5–26
5.5.8 Video and Graphics Control Register (VGR) 5–29
5.5.8.1 Video Display Pointer 5–31
5.5.9 Plane Mask Register (PLM) 5–32
5.5.10 Foreground Register (FOR) 5–32
5.6 Memory Controller Pins 5–33
5.6.1 mem_data<63:0> 5–33
5.6.2 mem_ecc<7:0> 5–34
5.6.3 mem_addr<11:0> 5–34
5.6.4 mem_write_l 5–34
5.6.5 mem_rasa_l<3:0> and mem_rasb_l<3:0> 5–35
5.6.6 mem_cas_l 5–35
5.6.7 mem_dtoe_l 5–35

Preliminary

5.6.8 mem_dsf 5–36
5.6.9 mem_rd_oe 5–36
5.6.10 mem_wr_oe_l 5–36
5.6.11 bc_tag<7:0> 5–36
5.6.12 bc_dirty 5–37
5.6.13 bc_parity 5–37
5.6.14 bc_index 5–37
5.6.15 bc_idx_tag<4:0> 5–37
5.6.16 bc_cs_l 5–38
5.6.17 bc_we_l 5–38
5.6.18 bc_oe_l 5–38
5.6.19 vframe_l 5–38
5.6.20 vrefresh_l 5–38
5.7 Memory Cycles 5–39
5.7.1 RAS Precharge Timing 5–43
5.7.2 DRAM Refresh Timing 5–44
5.7.3 Full Video Shift Register Transfer Timing 5–45
5.7.4 Split Video Shift Register Transfer Timing 5–46
5.7.5 Bcache Read Timing 5–47
5.7.6 Bcache Write Timing 5–48
5.7.7 Page Mode Read Timing 5–49
5.7.8 Nonpage Mode Read Timing 5–50
5.7.9 Page Mode Write Timing 5–51
5.7.10 Nonpage Mode Write Timing 5–52

6 I/O Controller

6.1 Introduction 6–1
6.1.1 Scatter Gather Mapping Support 6–1
6.1.2 Sparse and Dense Memory Support 6–2
6.1.3 PCI Parity Support 6–2
6.1.4 Exclusive Access Support 6–3
6.1.5 Clocking 6–3
6.2 CPU Initiated PCI Cycles 6–3
6.2.1 CPU Request Queue 6–4
6.2.2 Requesting PCI Mastership 6–5
6.2.3 Default Mastership (Bus Parking) 6–5
6.2.4 Memory Barrier Instruction Requirements 6–6
6.2.5 Address Phase 6–6

Preliminary

6.2.6 Sparse Memory Space 6–6
6.2.6.1 Generating PCI Addresses for Sparse Space 6–7
6.2.6.2 Address Extension For Sparse Space 6–8
6.2.6.3 Burst Order in Sparse Space 6–9
6.2.6.4 Unmasked Longword and Quadword Reads to Sparse Space 6–9
6.2.6.5 Unmasked Longword Writes to Sparse Space 6–10
6.2.6.6 Unmasked Quadword Writes to Sparse Space 6–10
6.2.6.7 Masked Longword Reads to Sparse Space 6–11
6.2.6.8 Masked Longword Writes to Sparse Space 6–11
6.2.7 Dense Memory Space 6–11
6.2.7.1 Generating PCI Addresses for Dense Space 6–11
6.2.7.2 Unmasked Reads 6–12
6.2.7.3 Unmasked Burst Writes 6–12
6.2.8 I/O Space Cycles 6–13
6.2.9 Configuration Cycles 6–13
6.2.10 Special Cycles and Interrupt Acknowledge Cycles 6–14
6.2.11 Burst Order 6–15
6.2.12 Back-to-Back Transfers 6–15
6.2.13 PCI Address Parity 6–16
6.2.14 PCI Data Parity 6–16
6.2.14.1 PCI Read Data Parity Errors 6–16
6.2.14.2 PCI Write Data Parity Errors 6–17
6.2.15 PCI Master Timeout 6–17
6.2.16 PCI Target Disconnects 6–17
6.2.17 PCI Retry Limit 6–18
6.2.18 PCI Target Abort 6–18
6.2.19 PCI Master Abort 6–19
6.3 Peripheral Initiated PCI Cycles 6–19
6.3.1 PCI Cycle Command Code Support and Aliasing 6–20
6.3.2 PCI Address Translation 6–20
6.3.2.1 Direct Mapped Address Translation 6–22
6.3.2.2 Scatter Gather Mapped Address Translation 6–23
6.3.3 Scatter Gather Translation Buffer 6–25
6.3.4 Page Table Read Errors 6–26
6.3.5 Invalid Page Errors 6–27
6.3.6 Address Phase 6–27
6.3.6.1 PCI Address Queue 6–27
6.3.6.2 Address Burst Order 6–28
6.3.6.3 Page Boundary Crossing 6–28
6.3.6.4 Address Parity Errors 6–28

Preliminary

6.3.7 PCI Initiated Write Data Transfers 6–29
6.3.7.1 PCI Write Data Queue 6–29
6.3.7.2 PCI Initiated Memory Write Data Errors 6–29
6.3.8 PCI Initiated Read Data Transfers 6–30
6.3.8.1 PCI Read Data Queue 6–31
6.3.8.2 PCI Initiated Memory Read Data Errors 6–31
6.3.9 PCI Target Latency Timeout 6–31
6.3.10 PCI Initiated Exclusive Access 6–32
6.3.11 Guaranteed Access Arbitration 6–32
6.4 CPU Access of IOC Registers 6–33
6.5 IOC Registers 6–33
6.5.1 Host Address Extension Register (IOC_HAE) 6–33
6.5.2 Configuration Cycle Type Register (IOC_CONF) 6–34
6.5.3 Translation Buffer Enable Register (IOC_TB_ENA) 6–35
6.5.4 PCI Soft Reset Register (IOC_SFT_RST) 6–36
6.5.5 PCI Parity Disable Register (IOC_PAR_DIS) 6–37
6.5.6 Status 0 Register (IOC_STAT0) 6–38
6.5.7 Status 1 Register (IOC_STAT1) 6–40
6.5.8 Translation Buffer Invalidate All Register (IOC_TBIA) 6–41
6.5.9 PCI Target Window Registers 6–42
6.5.9.1 Window Base Registers (IOC_W_BASE<1:0>) 6–43
6.5.9.2 Window Mask Registers (IOC_W_MASK<1:0>) 6–44
6.5.9.3 Translated Base Registers (IOC_T_BASE<1:0>) 6–44
6.5.10 Translation Buffer Tag Registers (IOC_TB_TAG<7:0>) 6–45
6.5.11 Interrupt Vector and Special Cycle Register (IOC_IACK_SC) 6–47
6.5.12 Initialization Requirements 6–48
6.6 IOC Signal Pins 6–48
6.6.1 ad<31:0> 6–48
6.6.2 c_be_l<3:0> 6–48
6.6.3 frame_l 6–48
6.6.4 trdy_l 6–49
6.6.5 irdy_l 6–49
6.6.6 stop_l 6–49
6.6.7 par 6–49
6.6.8 perr_l 6–49
6.6.9 devsel_l 6–49
6.6.10 lock_l 6–50
6.6.11 req_l 6–50
6.6.12 gnt_l 6–50
6.6.13 rst_l 6–50
6.6.14 pci_clk_in 6–51
6.6.15 memreq_l 6–51

Preliminary

6.6.16 memack_l 6–51

7 Clocks

7.1 Introduction 7–1
7.2 Clock Pins 7–1
7.2.1 pll_clk_in, pll_clk_in_l 7–1
7.2.2 pll_bypass 7–2
7.2.3 pll_i_ref 7–2
7.2.4 pll_filter 7–2
7.2.5 pll_5v 7–2
7.2.6 test_clk_out 7–3
7.2.7 reset_in_l 7–3
7.2.8 irq<2:0> 7–3
7.3 PLL System Overview 7–3

8 JTAG

8.1 JTAG Test Port 8–1
8.2 JTAG Pins 8–1
8.3 TAP Controller 8–1
8.4 Instruction Register 8–1
8.5 Bypass Register 8–2
8.6 Boundary Scan Register 8–3
8.6.1 Boundary Scan Register Cells 8–3
8.6.2 Boundary Scan Register Organization 8–4
8.7 Initialization 8–5

9 Icache Initialization and SROM Interface

9.1 Introduction 9–1
9.2 Instruction Cache Initialization/SROM Interface 9–1
9.3 Serial Line Interface 9–5

A Pin Summary

A.1 Memory Controller Pins A–1
A.2 PCI Bus Pins A–2
A.3 Clock Pins A–3
A.4 JTAG Pins A–3
A.5 Interrupt, SROM Interface, and INSTR_REF Pins A–4

Preliminary

B Internal Register Summary

B.1 Internal Processor Registers B–1
B.2 Memory Controller Registers B–3
B.3 I/O Controller Registers B–3

Index

Figures

2–1 Block Diagram 2–2
2–2 Instruction Pipelines 2–17
2–3 Producer-Consumer Latency Matrix 2–21
3–1 HW_MFPR and HW_MTPR Instruction Format 3–18
3–2 HW_LD and HW_ST Instructions Format 3–21
3–3 HW_REI Instruction Format 3–22
4–1 Translation Buffer Tag Register 4–2
4–2 Translation Buffer Page Table Entry Register 4–3
4–3 Instruction Cache Control and Status Register (ICCSR) 4–5
4–4 ITB_PTE_TEMP Register 4–10
4–5 Exception Address Register 4–11
4–6 Clear Serial Line Interrupt Register (SL_CLR) 4–12
4–7 Serial Line Receive Register 4–13
4–8 Processor Status Register 4–14
4–9 Exception Summary Register (EXC_SUM) 4–15
4–10 PAL Base Address Register 4–16
4–11 Hardware Interrupt Request Register (HIRR) 4–17
4–12 Software Interrupt Request Register 4–19
4–13 Asynchronous Trap Request Register 4–20
4–14 Hardware Interrupt Enable Register (HIER) 4–21
4–15 Software Interrupt Enable Register 4–23
4–16 AST Interrupt Enable Register 4–24
4–17 Serial Line Transmit Register 4–25
4–18 Translation Buffer Control Register 4–25
4–19 Data Translation Buffer Page Table Entry Register 4–26
4–20 Data translation Buffer Page table Entry Temporary Register 4–27

Preliminary

4–21	Memory Management Control and Status Register (MM_CSR)	4–28
4–22	Abox Control Register (ABOX_CTL)	4–29
4–23	Alternate Processor Mode Register (ALT_MODE)	4–32
4–24	Cycle Counter Register (Read)	4–33
4–25	Cycle Counter Control Register	4–34
4–26	Data Cache Status Register (DC_STAT)	4–34
5–1	Typical Memory System	5–2
5–2	ECC Code	5–5
5–3	Memory Controller Operation	5–8
5–4	Bank Configuration Register (BCR3–BCR0)	5–13
5–5	Row Address Bit Assignments	5–16
5–6	Bank Address Mask Register (BMR3–BMR0)	5–17
5–7	Bank Size Mask Setting	5–17
5–8	Bank Timing Register (BTR3–BTR0)	5–19
5–9	Global Timing Register (GTR)	5–20
5–10	Error Status Register (ESR)	5–22
5–11	Error Address Register (EAR)	5–25
5–12	Cache Register (CAR)	5–26
5–13	Cache Address Partition	5–29
5–14	Video and Graphics Control Register (VGR)	5–30
5–15	Plane Mask Register (PLM)	5–32
5–16	Foreground Register (FOR)	5–33
5–17	RAS Precharge Timing	5–43
5–18	DRAM Refresh Timing	5–44
5–19	Full Video Shift Register Transfer Timing	5–45
5–20	Split Video Shift Register Transfer Timing	5–46
5–21	Bcache Read Timing	5–47
5–22	Bcache Write Timing	5–48
5–23	Page Mode Read Timing	5–49
5–24	Nonpage Mode Read Timing	5–50
5–25	Page Mode Write Timing	5–51
5–26	Nonpage Mode Write Timing	5–52
6–1	CPU-to-PCI Address Translation	6–4
6–2	PCI Window Mask	6–21
6–3	PCI Address Translation - Scatter Gather Mapping Disabled	6–23

Preliminary

6–4 Scatter Gather Map Address 6–24
6–5 Scatter Gather Map Page Table Entry 6–25
6–6 Host Address Extension Register (IOC_HAE) 6–34
6–7 Configuration Cycle Type Register (IOC_CONF) 6–35
6–8 Translation Buffer Enable Register (IOC_TB_ENA) 6–35
6–9 PCI Soft Reset Register (IOC_SFT_RST) 6–36
6–10 PCI Parity Disable Register (IOC_PAR_DIS) 6–37
6–11 Status 0 Register (IOC_STAT0) 6–38
6–12 Status 1 Register (IOC_STAT1) 6–41
6–13 Translation Buffer Invalidate All Register (IOC_TBIA) 6–42
6–14 Window Base Registers (IOC_W_BASE<1:0>) 6–43
6–15 Window Mask Registers (IOC_W_MASK<1:0>) 6–44
6–16 Translated Base Registers (IOC_T_BASE<1:0>) 6–45
6–17 Translation Buffer Tag Registers (IOC_TB_TAG<7:0>) 6–46
6–18 Interrupt Vector/Special Cycle Register (IOC_IACK_SC) 6–47
7–1 Reference Resistor and Filter Capacitor 7–2
7–2 PLL Clock Generator 7–4
9–1 Icache Load Order 9–3
9–2 SROM Timing 9–4

Tables

2–1 Producer-Consumer Classes 2–20
2–2 Opcode Summary with Instruction Issue Bus 2–24
2–3 Required PALcode Instructions 2–25
2–4 Implementation-Specific Instructions 2–25
2–5 Physical Address Space 2–27
3–1 PALcode Entry Points 3–6
3–2 D-stream Error PAL Entry Points 3–7
3–3 HW_MTPR Restrictions 3–11
3–4 HW_MTPR Cycle Delay 3–14
3–5 HW_MFPR and HW_MTPR Format Description 3–19
3–6 Internal Processor Register Access 3–19
3–7 HW_LD and HW_ST Format Description 3–21
3–8 HW_REI Format Description 3–22
4–1 Instruction Cache Control and Status Register (ICCSR) Bit Description 4–5

Preliminary

4–2	BHE, BPE Branch Prediction Selection	4–7
4–3	Counter 0 Input Selection	4–8
4–4	Counter 1 Input Selection	4–8
4–5	Clear Serial Line Interrupt Register (SL_CLR) Bit Description	4–12
4–6	Exception Summary Register (EXC_SUM) Bit Description	4–15
4–7	Hardware Interrupt Request Register (HIRR) Bit Description	4–17
4–8	Hardware Interrupt Enable Register (HIER) Bit Description	4–21
4–9	Memory Management Control and Status Register (MM_CSR) Bit Description	4–28
4–10	Abox Control Register (ABOX_CTL) Bit Description	4–30
4–11	Alternate Processor Mode Register (ALT_MODE) Bit Description	4–32
4–12	Data Cache Status Register (DC_STAT) Bit Description	4–35
4–13	Internal Process Register Reset State	4–35
5–1	Nibble Correction Bit Grouping	5–5
5–2	Memory Controller Registers	5–12
5–3	Bank Configuration Register (BCR3–BCR0) Bit Description	5–13
5–4	Row Address Select Encodings	5–15
5–5	RAS Selection	5–15
5–6	Bank Address Mask Register (BMR3–BMR0) Bit Description	5–17
5–7	Split Bank Address Select	5–18
5–8	Bank Timing Register (BTR3–BTR0) Bit Description	5–19
5–9	Global Timing Register (GTR) Bit Description	5–20
5–10	Error Status Register (ESR) Bit Description	5–22
5–11	Error Address Register (ESR) Bit Description	5–25
5–12	Performance Count Events	5–25
5–13	Cache Register (CAR) Bit Description	5–27
5–14	External Cache Size Encodings	5–28
5–15	Video and Graphics Control Register (VGR) Bit Description	5–30
5–16	VRAM Address Increment Encodings	5–31
5–17	Plane Mask Register (PLM) Bit Description	5–32

Preliminary

5–18 Foreground Register (FOR) Bit Description 5–33
5–19 DSF Function 5–36
5–20 Use of bc_idx_tag 5–37
5–21 Memory Timing Parameters 5–39
5–22 Memory Operations 5–40
5–23 Memory Read/Write Options 5–42
6–1 CPU Address Windows to PCI Address Spaces 6–4
6–2 Byte Enable Encoding for CPU Initiated Transfers in Sparse Space 6–7
6–3 Host Address Extension 6–8
6–4 Supported Commands for PCI Initiated Transactions 6–20
6–5 Unsupported Commands for PCI Initiated Transfers 6–20
6–6 IOC Register Bit Access Behavior 6–33
6–7 Host Address Extension Register (IOC_HAE) Bit Description 6–34
6–8 Configuration Cycle Type Register (IOC_CONF) Bit Description 6–35
6–9 Translation Buffer Enable Register (IOC_TB_ENA) Bit Description 6–36
6–10 PCI Soft Reset Register (IOC_SFT_RST)Bit Description 6–36
6–11 PCI Parity Disable Register (IOC_PAR_DIS) Bit Description 6–37
6–12 Status 0 Register (IOC_STAT0) Bit Description 6–39
6–13 Status 1 Register (IOC_STAT1) Bit Description 6–41
6–14 Translation Buffer Invalidate All Register (IOC_TBIA) Bit Description 6–42
6–15 PCI Target Window Register Addresses 6–42
6–16 Window Base Registers (IOC_W_BASE<1:0>) Bit Description 6–43
6–17 Window Mask Registers (IOC_W_MASK<1:0>) Bit Description 6–44
6–18 Translated Base Registers (IOC_T_BASE<1:0>) Bit Description 6–45
6–19 TLB Tag Register Addresses 6–45
6–20 Translation Buffer Tag Registers (IOC_TB_TAG<7:0>) Bit Description 6–46
6–21 Interrupt Vector/Special Cycle Register (IOC_IACK_SC) Bit Description 6–47

Preliminary

7–1	Internal Clock programming	7–5
8–1	JTAG Instruction Register	8–2
8–2	JTAG ICMODE Functions	8–3
8–3	Boundary Scan Order	8–4
9–1	Icache Test Modes	9–1
A–1	Memory Controller Pins	A–1
A–2	PCI Bus Signal Pins	A–2
A–3	Clock Pins	A–3
A–4	JTAG Pins	A–3
A–5	Interrupt, SROM Interface, and INSTR_REF Pins	A–4
B–1	Implementation-Specific Internal Processor Registers	B–1
B–2	Memory Controller Registers	B–3
B–3	I/O Controller Registers	B–4

Preliminary

Preface

This manual describes the architecture, internal design, and external interface of the DECchip 21066 and DECchip 21068 microprocessors. The physical, electrical, and thermal characteristics are descibed in the *DECchip 21066 Microprocessor Data Sheet* and the *DECchip 21068 Microprocessor Data Sheet*.

Audience

This manual is for system designers, software developers, and hardware engineers who use the DECchip 21066 and DECchip 21068 microprocessors.

Structure

This manual contains the following chapters and appendices and an index.

- Chapter 1 Introduction
- Chapter 2 Internal Architecture
- Chapter 3 Priviliged Architecture Library Code
- Chapter 4 Internal Processor Registers
- Chapter 5 Memory Controller
- Chapter 6 I/O Controller
- Chapter 7 Clocks
- Chapter 8 JTAG
- Chapter 9 Icache Initialization and SROM Interface
- Appendix A Pin Summary
- Appendix B Internal Register Summary

Preliminary

Preliminary

Conventions

Unless otherwise noted, the following conventions are extracted from the *Alpha Architecture Reference Manual.* The most recent version of the *Alpha Architecture Reference Manual* is the authoritative reference for such information, and may supersede information here.

Addresses

In figures, tables, and examples, addresses increase from right to left and bottom to top.[1]

Aligned and Unaligned

The terms *aligned* and *naturally aligned* are interchangeable and refer to data objects that are powers of two in size. An aligned datum of size 2^n is stored in memory at a byte address that is a multiple of 2^n; that is, one that has n low-order zeros. For example, an aligned 64-byte stack frame has a memory address that is a multiple of 64.

A datum of size 2^n is *unaligned* if it is stored in a byte address that is not a multiple of 2^n.

Bit and Field Abbreviations

IGN — Ignore

Bits and fields specified as IGN are ignored when written.

IMP — Implementation Dependent

Bits and fields specified as IMP may be used for implementation-specific purposes. Each implementation must document fully the behavior of all fields specified as IMP by the Alpha specification.

MBZ — Must Be Zero

[1] Implementation-specific or typographical convention adapted from or not specified in the *Alpha Architecture Reference Manual.*

Preliminary

Software must never place a nonzero value in bits and fields specified as MBZ. Such fields are reserved for future use. An illegal operand exception occurs if the processor encounters a nonzero value in a field specified as MBZ.

RAS — Read As Sign

Bits and fields specified as RAS are unimplemented bits of registers that read back as copies of the most significant implemented bit.[1]

RES — Reserved

Bits and fields specified as RES are reserved by Digital and should not be used.[1]

RAX — Read As UNDEFINED

Bits and fields specified as RAX are unimplemented bits of registers that read back as UNDEFINED.[1]

RAZ — Read As Zero

Bits and fields specified as RAZ return a zero when read.

RWM — Read/Write/Modify

Bits and fields specified as RWM can be read and written. They can also be modified by hardware.[1]

RO — Read Only

Bits and fields specified as RO can be read but not written.[1]

RW — Read/Write

Bits and fields specified as RW can be read and written.[1]

R/W1C — Read/Write One to Clear

Bits and fields specified as R/W1C can be read. Writing a one clears the bits.[1]

SBZ — Should Be Zero

Software should place a zero value in bits and fields specified as SBZ. Nonzero values in such fields produce UNPREDICTABLE results and may produce extraneous instruction-issue delays.

WO — Write Only

Preliminary

Bits and fields specified as WO can be written but not read.[1]

Bit Notation

Multiple bit fields are shown as extents (see Ranges and Extents, below).(<>).[1]

Caution

Cautions indicate potential damage to equipment or data.[1]

Data Units

The following data unit terminology is used throughout this manual.

Term	Words	Bytes	Bits	Other
Nibble	¼	½	4	
Byte	½	1	8	
Tribyte	1½	3	24	
Word	1	2	16	
Longword	2	4	32	
Quadword	4	8	64	
Octaword	8	16	128	Single read fill; that is, the cache space that can be filled in a single read access. It takes two read accesses to fill a backup cache (Bcache) line.
Hexaword	16	32	256	Cache block, cache line. The space allocated to a single Bcache block.

Note

Notes indicate general information.[1]

Numbering

All numbers are decimal or hexadecimal unless otherwise indicated. In cases of ambiguity, a subscript indicates the radix of nondecimal numbers. For example, 19 is decimal, but 19_{16} and 19A are hexadecimal.[1]

Ranges and Extents

Ranges are specified by a pair of numbers separated by two periods (..) and are inclusive. For example, a range of integers 0..4 includes the integers 0, 1, 2, 3, and 4.

Extents are specified by a pair of numbers in angle brackets (<>) separated by a colon (:) and are inclusive. For example, bits <7:3> specifies an extent including bits 7, 6, 5, 4, and 3.

Preliminary

Security Holes

A security hole is an error of commission, omission, or oversight in a system that allows protection mechanisms to be bypassed.

Security holes exist when unprivileged software (that is, software running outside of kernel mode) can:

- Affect the operation of another process without authorization from the operating system.
- Amplify its privilege without authorization from the operating system.
- Communicate with another process, overtly or covertly, without authorization from the operating system.

The Alpha architecture is designed to contain no architectural security holes. Hardware (processors, buses, controllers, and so on) and software should also be designed to avoid security holes.

UNPREDICTABLE and UNDEFINED

Only privileged software (that is, software running in kernel mode) may trigger UNDEFINED operations.

Either privileged or unprivileged software may trigger UNPREDICTABLE results or occurrences.

UNPREDICTABLE results and occurrences do not disrupt the basic operation of the processor; the processor continues to execute instructions in its normal way.

UNDEFINED operation may halt the processor or cause it to lose information.

A result specified as UNPREDICTABLE may acquire an arbitrary value subject to a few constraints. Such a result may be an arbitrary function of the input operands or of any state information that is accessible to the process in its current access mode. UNPREDICTABLE results may be unchanged from their previous values. Operations that produce UNPREDICTABLE results may also produce exceptions.

UNPREDICTABLE results must not be security holes.

Specifically, UNPREDICTABLE results must not depend on or be a function of the contents of memory locations or registers that are inaccessible to the current process in the current access mode.

Also, operations that may produce UNPREDICTABLE results must not:

- Write or modify the contents of memory locations or registers which are inaccessible to the current process in the current access mode.

Preliminary

- Halt or hang the system or any of its components.

For example, a security hole would exist if some UNPREDICTABLE result depended on:

- the value of a register in another process,
- the contents of processor temporary registers left behind by some previously running process, or
- a sequence of actions of different processes.

An occurrence specified as UNPREDICTABLE may happen or not based on an arbitrary choice function. The choice function is subject to the same constraints as are UNPREDICTABLE results and, in particular, must not constitute a security hole.

Results or occurrences specified as UNPREDICTABLE may vary from moment to moment, implementation to implementation, and instruction to instruction within implementations. Software can never depend on results specified as UNPREDICTABLE.

Operations specified as UNDEFINED may vary from moment to moment, implementation to implementation, and instruction to instruction within implementations. The operation may vary in effect from nothing to stopping system operation. UNDEFINED operations must not cause the processor to hang; that is, reach an unhalted state from which there is not transition to a normal state in which the machine executes instructions.

Preliminary

Preliminary

1 Introduction

This chapter gives an overview of the Alpha AXP architecture as it is implemented in the DECchip 21066 and DECchip 21068 microprocessors. It also lists the major features of the microprocessors.

Reader's Note

Unless otherwise stated, throughout this manual the terms microprocessor and chip mean the DECchip 21066 or the DECchip 21068.

1.1 Architecture

The Alpha AXP architecture is a 64-bit load/store RISC architecture designed with particular emphasis on speed, multiple instruction issue, multiple processors, and software migration from OpenVMS and MIPS/ULTRIX.

All registers are 64 bits in length and all operations are performed between 64-bit registers. All instructions are 32 bits in length. Memory operations are either loads or stores. All data manipulation is done between registers.

The Alpha AXP architecture supports the following data types:

- 8-, 16-, 32- and 64-bit integers
- IEEE 32-bit and 64-bit floating point formats
- VAX computer 32-bit and 64-bit floating point formats

In the Alpha AXP architecture, instructions interact with each other only by one instruction writing to a register or memory location and another instruction reading from that register or memory location. This use of resources makes it easy to build implementations that issue multiple cycles every CPU cycle.

Preliminary

The microprocessor uses a set of subroutines, called privileged architecture library code (PALcode), that is specific to a particular Alpha AXP operating system implementation and hardware platform. These subroutines provide operating system primitives for context switching, interrupts, exceptions, and memory management. These subroutines can be invoked by hardware or CALL_PAL instructions. CALL_PAL instructions use the function field of the instruction to vector to a specified subroutine. PALcode is written in standard machine code with some implementation-specific extensions to provide direct access to low-level hardware functions. PALcode supports optimizations for multiple operating systems, flexible memory management implementations, and multi-instruction atomic sequences.

The Alpha AXP architecture does byte shifting and masking with normal 64-bit register-to-register instructions; it does not include single byte load/store instructions. The software implementor must determine the precision of arithmetic traps.

For a complete introduction to the Alpha AXP architecture, see the companion volume, the Alpha Architecture Handbook.

1.2 Chip Features

The microprocessor implements the Alpha AXP architecture. It is a 0.65 micron CMOS-based, superscalar, superpipelined processor using dual instruction issue. The chip incorporates a high level of system integration to provide best-in-class system performance for cost-focused (DECchip 21066) or embedded (DECchip 21068) applications. It integrates on-chip fully pipelined integer and floating-point processors, a high bandwidth memory controller, an industry standard I/O controller, an embedded graphics accelerator, internal instruction and data caches, and external cache control.

The microprocessor and associated PALcode implement IEEE single and double precision, VAX F_floating and G_floating data types and supports longword (32-bit) and quadword (64-bit) integers. Byte (8-bit) and word (16-bit) support is provided by byte manipulation instructions. Limited hardware support is provided for the VAX D_floating data type.

Summary of the microprocessor features:

- Fully pipelined, 64-bit RISC architecture.
- Supported by multiple operating systems:
  - Microsoft Windows/NT
  - OSF/1
  - OpenVMS

Preliminary

  - DECelx
- Best-in class performance:
  - 166 MHz operation
  - Superscalar superpipelined (dual issue).
  - Peak instruction execution rate of 330 million instructions per second (MIPS)
  - 0.65 micron CMOS technology.
- Pipelined on-chip floating-point unit:
  - IEEE single- and double-precision, DEC F_floating and G_floating, longword and quadword data types. Limited support for DEC D_ floating data type.
- On-chip demand-paged memory management Unit:
  - A 12-entry I-stream translation buffer with eight entries for 8KB pages and four entries for 4MB pages.
  - A 32-entry D-stream translation buffer with each entry able to map a single 8KB, 64KB, 512KB, or 4MB page.
  - Superpage mapping.
- On-chip high bandwidth memory controller:
  - Full 64-bit memory datapath.
  - Dynamic RAM (DRAM) controller with fully programmable timing.
  - Optional error correction code (ECC).
  - Supports up to 4 banks of memory with programmable timing for each bank.
  - RAS/CAS memory bus to industry standard single in-line memory modules (SIMMs).
- Embedded graphics accelerator:
  - Provides on-chip support for direct connection to Video RAM (VRAM) frame buffer.
  - High-bandwidth memory datapath to VRAM.
- PCI I/O controller:
  - 32-bit multiplexed address/data.

Preliminary

  - Industry standard.
  - No glue logic needed to connect PCI-compliant chips.
  - Usable bandwidth in excess of 100M bytes/s.
  - Burst mode reads and writes.
  - Asynchronous operation to CPU.
  - Multimaster with peer-to-peer access.
- On-chip 8KB direct mapped write-through data cache.
- On-chip 8KB direct mapped instruction cache.
- On-chip control for optional, external, write-back backup cache (Bcache):
  - Provides direct connection to external static RAMs (SRAMs) without any external logic.
  - Programmable cache size and speed.
- Built-in phase-locked loop (PLL)
  - Frequency multiplier allows low cost input clock.
  - Programmable multiplier values
- Serial ROM (SROM) interface:
  - Loads Icache after reset.
  - Software controlled serial port after initialization.
- Chip- and module-level test support:
  - JTAG (IEEE 1149).
- 3.3 V supply voltage
  - Interfaces directly to 5 V logic.

Preliminary

2
Internal Architecture

This chapter gives a programmer's view of the chip's microarchitecture. The chapter describes the hardware with minimal forward references to the pipeline described in Section 2.9). The scheduling and dual issue rules are defined Section 2.10 (the CPU can issue two instructions in a single cycle). This chapter is not a detailed hardware description of the chip. The combination of microarchitecture and PALcode defines the chip's implementation of the Alpha AXP architecture. Much of the hardware design is based on specific PAL functionality (PAL is described in Chapter 3). Figure 2–1 is a block diagram of the chip.

2.1 Overview

The chip consists of a core central processing unit (CPU), a memory controller, and an I/O controller (IOC). The chip also contains instruction and data caches and a serial read-only memory (SROM) interface.

The memory controller interfaces to the system memory and an optional, external, backup cache. It also contains the embedded graphics accelerator. The memory controller and interface are described in Chapter 5.

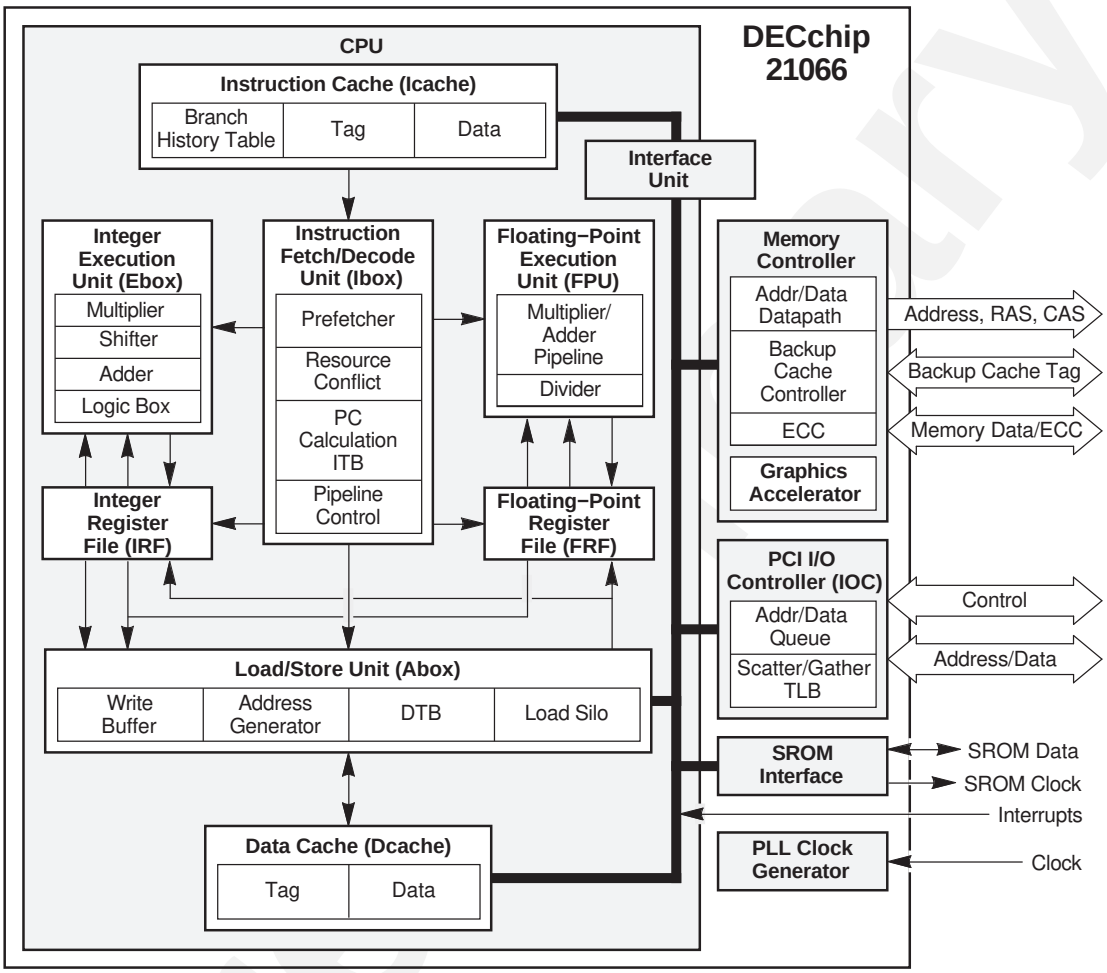
The PCI I/O controller (IOC) is an interface between peripheral devices and the CPU and system memory. It is compatible with the *PCI Local Bus Specification, Revision 2.0*. The IOC and interface are described in Chapter 6.

The SROM interface provides the initialization data load path from the SROM to the instruction cache. Following initialization, this interface can be used as a diagnostic port using PALcode. The SROM interface and operation are descibed in Chapter 9.

The interface unit (Figure 2–1) connects the CPU, memory controller, IOC, and the SROM interface. It consists of a 64-bit bidirectional data bus, address bus, invalidate address bus, reset logic, and control.

Preliminary

Figure 2–1 Block Diagram



The instruction fetch and decode unit (Ibox) is the CPU's central control unit. It issues instructions, maintains the pipeline, and performs program counter (PC) calculations. The CPU also contains four independent execution units:

- Integer execution unit (Ebox)
- Load and store unit (Abox)
- Floating-point unit (FPU)
- Branch unit

Preliminary

Each unit accepts no more than one instruction per cycle; however, correctly scheduled code can issue two instructions to two independent units in a single cycle.

2.2 Instruction Fetch and Decode Unit

The primary function of the instruction fetch and decode unit (Ibox) is to issue instructions to the integer execution unit, load and store unit, and floating-point unit. The Ibox contains:

- Prefetcher
- PC pipeline
- Two instruction translation buffers (ITBs)
- Abort logic
- Register conflict or dirty logic
- Exception logic
- Internal processor registers

2.2.1 Instruction Fetch and Decode

The Ibox decodes two instructions in parallel and checks that the required resources are available for both instructions, as follows:

- If resources are available, then both instructions are issued.
- If resources are available only for the second instruction, then neither instruction is issued.
- If the Ibox issues only the first of a pair of instructions, then it does not advance another instruction to attempt another dual issue; dual issue is attempted only on aligned quadword (8-byte) pairs.

Section 2.10.4 describes instructions that can be dual-issued.

2.2.2 Branch Prediction Logic

The branch unit, or prediction logic, is also part of the Ibox. The microprocessor offers a register-selectable choice of branch prediction strategies (see the instruction cache control and status register, ICCSR, Section 4.2.3). Each instruction location in the instruction cache includes a single history bit to record the outcome of branch instructions. This information can be used to predict the result when the branch instruction is next executed. The prediction for the first execution of a branch instruction is based on the sign of the displacement field within the branch instruction itself.

Preliminary

- If the sign bit is negative, the instruction prefetcher predicts the conditional branches to be taken.
- If the sign is positive, the instruction prefetcher predicts the conditional branches not to be taken.
- Alternatively, if the history table is disabled, branches can be predicted at all times based on the sign of the displacement field.

The microprocessor provides a four-entry subroutine return stack that is controlled by the hint bits in the BSR, HW_REI, JMP, JSR, RET, and JSR_COROUTINE instructions. The chip also allows all branch prediction hardware to be disabled.

2.2.3 Instruction Translation Buffers

The Ibox includes two fully-associative instruction translation buffers (ITBs):

- An 8-entry small-page ITB for 8KB pages.
- A 4-entry large-page ITB that supports 512 8KB pages.

Both translation buffers store recently used, instruction stream (I-stream) page table entries (PTEs) and use a not-last-used replacement algorithm.

In addition, both ITBs support a register-enabled extension called the superpage. The ITB superpage mappings provide one-to-one virtual program counter <33:13> to physical program counter <33:13> translation when virtual address bits <42:41> = 2. When translating through the superpage, the page-table entry (PTE) address-space match (ASM) bit used in the instruction cache (Icache) is always set. Access to the superpage mapping is allowed only while executing in kernel mode.

PALcode fills and maintains the ITBs. The operating system, through PALcode, is responsible for ensuring that virtual addresses can only be mapped through a single ITB entry (in the large page, small page, or superpage) at the same time.

The Ibox presents the 43-bit virtual program counter (VPC) to the ITB each cycle while not executing in PALmode. If the PTE associated with the VPC is cached in the ITB then the Ibox uses the page frame number (PFN) and protection bits for the page that contains the VPC to complete the address translation and access checks.

The ITBs support a single address space number (ASN) through the PTE address space match (ASM) bit. Each PTE entry in the ITB contains an ASM bit. Writes to the instruction translation buffer ASM register (ITBASM, Section 4.2.9) invalidate all entries that do not have the ASM bit set. This

Preliminary

is a simple way to preserve entries that map operating system regions while invalidating all other entries.

For more information about translation buffers and page table entries, see the *Alpha Architecture Reference Manual*.

2.2.4 Interrupt Logic

The Ibox exception logic supports three sources of interrupts:

- Hardware interrupts
  - There are three level-sensitive hardware interrupts sourced by pins **irq<2:0>**.
  - There are two internally-generated interrupts that respond to external interface error conditions. These are sourced by the memory controller error status register (ESR, Section 5.5.5) and the I/O controller status register (IOC_STAT0, Section 6.5.6).
- Software interrupts

  There are 15 prioritized software interrupts, sourced by the software interrupt request register (SIRR, Section 4.2.15).
- Asynchronous system traps (ASTs)

  There are four ASTs, one for each processor mode: user, supervisor, executive, and kernel. These traps are sourced by the asynchronous trap request register (ASTRR, Section 4.2.16).

In addition to the three level-sensitive hardware interrupts from the external pins, error conditions detected in the memory controller and the I/O controller can interrupt the CPU.

The memory controller generates an interrupt if it detects any error condition, such as nonexistent memory access, missed refresh cycle, uncorrectable ECC error, backup cache parity error, and so on. Software must clear these error conditions by writing to the memory controller ESR to return the memory system to proper operation.

The I/O controller generates an interrupt to the CPU when it detects certain error conditions. These errors are recorded in the IOC_STAT0 register. Software must write the logged error bit in the IOC_STAT0 register to remove the interrupt.

The memory controller and IOC error conditions and registers are described in Chapters 5 and 6.

Preliminary

All interrupts can be independently masked by on-chip enable registers. In addition, AST interrupts are qualified by the current processor mode and the current state of software interrupt enable register bit 2 (SIER, Section 4.2.18). By providing distinct enable bits for each independent interrupt source, a software controlled interrupt priority scheme can be implemented by PALcode or the operating system with maximum flexibility.

For example, the microprocessor can support a 5-level interrupt priority scheme. This is done by defining a distinct state of the hardware interrupt enable register (HIER, Section 4.2.17) for each interrupt priority level (IPL). The state of the HIER determines the current interrupt priority. The lowest IPL is produced by enabling all five interrupts, for example bits <5:1>. The next IPL is produced by enabling bits <5:2> and so on, to the highest IPL that is produced by enabling only bit 5, and disabling bits <4:1>.

When all interrupt enable bits are cleared, the processor cannot be interrupted from the hardware interrupt request register (HIRR, Section 4.2.14). Each state (<5:1> enabled through bit 5 only enabled) represents an individual IPL. If these states are the only states allowed in the HIER, a three-level hardware interrupt priority scheme can be controlled entirely by PAL software. The scheme is extensible to provide multiple interrupt sources at the same interrupt priority level by grouping enable bits. Groups of enable bits must be set and cleared together to support multiple interrupts of equal priority level. This method reduces the total available number of distinct levels.

Because enable bits are provided for all hardware, software, and AST interrupt requests, a priority scheme can span all sources of processor interrupts. The only exception to this rule is the following restriction on AST interrupt requests:

- Four AST interrupts are provided, one for each processor mode. AST interrupt requests are qualified such that AST requests corresponding to a given mode are blocked when the processor is in a higher mode, regardless of the state of the AST interrupt enable register (ASTER, Section 4.2.19). In addition, all AST interrupt requests are qualified by SIER bit 2.
- When the processor receives an interrupt request and:
  - that request is enabled and
  - the processor is not currently executing PALcode

  the hardware reports or delivers an interrupt to the exception logic. Before vectoring to the interrupt-service PAL dispatch address, the pipeline is completely drained and all outstanding data cache fills are completed. The restart address is saved in the exception address register (EXC_ ADDR, Section 4.2.5) and the processor enters PALmode. The cause of

Preliminary

the interrupt can be determined by examining the state of the interrupt request registers.

In typical operation, the interrupts from the three external pins can be grouped as follows:

- Interval timer interrupts
- Halt, power fail, ISA/EISA nonmaskable interrupts, and PCI **serr_1**
- ISA/EISA Interrupt requests, and PCI Interrupt requests

Hardware interrupt requests are level-sensitive; therefore, they may be removed before an interrupt is serviced. If they are removed before the interrupt request register is read, the register will return a zero value.

2.2.5 Performance Counters

An on-chip performance recording mechanism counts various hardware events and causes an interrupt upon counter overflow. Interrupts are triggered six cycles after the event, and therefore, the exception program counter may not reflect the exact instruction causing counter overflow.

Two counters are provided, to allow accurate comparison of two variables under potentially nonrepeatable, experimental conditions. The events counted include:

- Instruction issues
- Instruction nonissues
- Total cycles
- Pipeline dry
- Pipeline freeze
- Brance mispredictions and cache misses
- Counts of various instruction classes

In addition, two external interface events (such as DMA transactions, external cache accesses, and so on) can be counted by programming the memory controller error address register (EAR, Section 5.5.6).

Preliminary

2.3 Integer Execution Unit

The integer execution unit (Ebox) contains the 64-bit integer execution data path, which includes the following:

- Adder
- Logic box
- Barrel shifter
- Byte zapper
- Bypassers
- Integer multiplier

The integer multiplier retires four bits per cycle.

The Ebox also contains the 32-entry, 64-bit integer register file (IRF). The IRF has four read ports and two write ports to simultaneously read operands to and write operands (results) from the integer execution data path and the load and store unit.

2.4 Load and Store Unit

The load and store unit (Abox) contains four major sections:

- Address translation data path, which includes the data translation buffer
- Load silo
- Write buffer
- Internal processor registers (IPRs)

The address translation datapath has a displacement adder that generates the effective virtual address for load and store instructions, and a data translation buffer (DTB) that generates the corresponding physical address.

The CPU requests fall into three classes: data cache (Dcache) fills, Icache fills, and write buffer requests. Simultaneous internal requests are resolved using a fixed priority scheme in which Dcache fill requests are given highest priority, followed by Icache fill requests. Write buffer requests have the lowest priority.

Because the internal cache fill block size is 32 bytes, cache fill operations result in four data transfers across a 64-bit bus from the memory controller to the appropriate cache. Also, because each write buffer is 32 bytes wide, write transactions may result in four data transfers from the write buffer to the memory controller.

The internal processor registers are described in Chapter 4.

Preliminary

2.4.1 Data Translation Buffer (DTB)

The 32-entry, fully-associative, data translation buffer (DTB) stores recently used, data stream (D-stream) page table entries (PTEs). The DTB supports four page-size granularity options (also called granularity hints) that allow an aligned group of 1, 8, 64, or 512 pages to be be treated as a single larger page.

The DTB also supports the register-enabled superpage extension. The DTB superpage mappings provide virtual-to-physical address translation for two regions of the virtual address space:

- The first enables superpage mapping when virtual address bits <42:41> = 2. In this mode, the entire physical address space is mapped multiple times to the quadrant of virtual address space defined by VA <42:41> = 2.
- The second maps a 30-bit region of the total physical address space, defined by PA <33:30> = 0, into a single corresponding region of virtual address space defined by VA <42:30> = 1FFE.

Superpage translation is allowed only in kernel mode. The operating system, through PALcode, is responsible for ensuring that translation buffer entries, including superpage regions, do not map overlapping virtual address regions at the same time.

Each PTE in the DTB contains an address space match (ASM) bit. The DTB supports a single address space number (ASN) with the PTE ASM bit. Writes to the data translation buffer ASM register (DTBASM, Section 4.3.7) invalidate all entries that do not have the ASM bit set. This is a simple way to preserve entries that map operating system regions while invalidating all other entries.

For load and store instructions, the effective 43-bit virtual address is presented to the DTBs. If the PTE of the supplied virtual address is cached in the DTB, the Abox uses the page frame number (PFN) and protection bits for the page that contains the address to complete the address translation and access checks.

PALcode fills and maintains the DTB. The flow for a DTB miss is described in Chapter 3. Note that the DTB can be filled in kernel mode by setting the HWE bit in the instruction cache control and status register (ICCSR, Section 4.2.3).

For more information about translation buffers and page table entries, see the *Alpha Architecture Reference Manual*.

Preliminary

2.4.2 Load Silos

The Abox contains a memory reference pipeline that can accept a new load or store instruction every cycle until a Dcache fill is required. Because the Dcache lines are allocated only on load misses, the Abox can accept a new load instruction every cycle until a load miss occurs. When a load miss occurs, the Ibox stops issuing all instructions that use the load port of the register file or are otherwise handled by the Abox. This includes LDx, STx, HW_MTPR, HW_MFPR, FETCH, FETCH_M, RPCC, RS, RC, and MB. It also includes all memory format branch instructions, JMP, JSR, JSR_COROUTINE, and RET. (However, a JSR with a destination of R31 can be issued.)

Instructions are issued in pipeline stage 3 and the result of each data cache lookup is not known until pipeline stage 6. Therefore, there can be two instructions in the Abox pipeline behind a load instruction that misses in the data cache. These two instructions are handled as follows:

- Loads that hit in the data cache are allowed to complete (hit-under-miss).
- Loads that miss are placed in a silo and are replayed in sequence after the first load miss completes.
- Store instructions are presented to the data cache at their normal time, with respect to the pipeline. They are placed in a silo and presented to the write buffer in sequence, with respect to loads that miss.

To improve performance, the Ibox can restart the execution of Abox-directed instructions before the last pending Dcache fill is complete. Dcache fill transactions result in four data transfers from the memory controller to the Dcache. These transfers can each be separated by one or more cycles, depending on the characteristics of the external cache and memory subsystems. In the first of these four transfers, the memory controller sends the quadword of the fill block originally requested by the CPU (it is always able to accomplish this for reads that hit in the external cache). Therefore, the pending load instruction that requested the Dcache fill can complete before the Dcache fill finishes.

Dcache fill data accumulates one quadword at a time, into a pending fill latch, rather than being written into the cache array as it is received from the memory controller. When the load miss silo is empty and the requested quadword for the last outstanding load miss is received, the Ibox resumes execution of Abox-directed instructions despite the still-pending Dcache fill. When the entire cache line has been received from the memory controller, it is written into the Dcache data array when the array is not busy with a load or a store.

Preliminary

2.4.3 Write Buffer

The Abox write buffer has two purposes:

- The CPU can generate store data faster than the backup cache (Bcache) subsystem can accept the data. This can cause CPU stall cycles. The write buffer provides a finite, high-bandwidth resource for receiving store data to minimize the number of possible CPU stall cycles.
- The write buffer also attempts to aggregate store data into aligned, 32-byte cache blocks to maximize the rate at which the microprocessor can write data into the Bcache.

The write-merging operation of the write buffer can cause the order of off-chip writes to be different from the order in which their corresponding store instructions were executed. In addition, the write buffer can collapse multiple stores to the same location into a single off-chip write transaction. Software that requires strict write ordering, or that does multiple stores to the same location (resulting in multiple off-chip write sequences), must insert a memory barrier instruction between the store instructions of interest.

In addition to store instructions, MB, STQ/C, STL/C, FETCH, and FETCH_M instructions are written into the write buffer and sent off chip. Unlike stores, however, these write buffer-directed instructions are never merged into a write buffer entry with other instructions.

The of the four write buffer entries can store up to 32 bytes. The buffer has a *head* pointer and *tail* pointer. The buffer puts new commands into empty tail entries and takes commands out of nonempty head entries. The head pointer increments when an entry is unloaded to the memory controller and the tail pointer increments when new data is put into the tail entry. The head and tail pointers point to the same entry only when the buffer contains zero or four nonempty entries.

for example, suppose that no writes merge with existing nonempty entries. In that case the ordering of writes with respect to other writes will be maintained. The write buffer never reorders writes except to merge them into nonempty entries. After write merges into a nonempty entry, its programmed order is lost with respect to writes in the same slot and writes in other slots.

The write buffer attempts to send its head entry off chip by requesting the memory controller when one of the following conditions is met:

- The write buffer contains at least two valid entries.
- The write buffer contains one valid entry and at least 256 CPU cycles have elapsed since the execution of the last write buffer-directed instruction.

Preliminary

- The write buffer contains an MB instruction.
- The write buffer contains an STQ/C or STL/C instruction.
- A load miss is pending to an address currently valid in the write buffer that requires the write buffer to be flushed. The write buffer is completely flushed regardless of which entry matches the address.

2.4.3.1 Memory Barrier Instruction

The CPU will not execute load or store instructions that occur subsequent to a memory barrier (MB) instruction until all load or store instructions preceeding the MB instruction have been completed. Such instructions are considered to be completed as follows:

- A load from a memory address completes when the memory controller returns the requested read data to the CPU.
- A load from a PCI address completes when the requested read data has been returned to the CPU.
- A store to a memory address completes when the write data has been written to memory.
- A store to a PCI address completes when the write data has been transfered to the PCI (and is not being retried).

A load or store is considered to be completed, whether or not it completes with error status. See Chapter 6 for more information.

2.5 Floating-Point Unit

The on-chip, pipelined floating-point unit (FPU) can execute both IEEE and VAX floating-point instructions. The microprocessor supports IEEE S_floating and T_floating data types, with all rounding modes (except round to ± infinity, which can be provided in software). The microprocessor fully supports VAX F_floating and G_floating data types, and provides limited support for the VAX D_floating format.

The FPU contains:

- A 32-entry, 64-bit floating-point register file (FRF)
- The user-accessible floating-point control register (FPCR), containing:
  - Round mode controls
  - Exception flag information

For a desciption of the FPCR, see the *Alpha Architecture Reference Manual*.

Preliminary

The FPU can accept an instruction every cycle, with the exception of floating-point divide instructions. The latency for data-dependent, nondivide instructions is six cycles. For more information about instruction timing, see Section 2.9.

For divide instructions, the FPU does not compute the inexact flag. Consequently, the INE exception flag in the FPCR register is never set for IEEE floating-point divide using the inexact enable (/I) qualifier. To deliver IEEE-compliant exception behavior, the FPU hardware always traps on DIVS /SI and DIVT/SI instructions. This allows the arithmetic exception handler (in either PALcode or the operating system) to identify the source of the trap, compute the inexact flag, and deliver the appropriate exception. The exception associated with DIV/SI and DIVT/SI is imprecise. Software must follow the rules specified by the Alpha AXP architecture associated with the software completion modifier in order to ensure that the trap handler can deliver correct behavior to the user.

For IEEE compliance issues, see Section 2.5.1 and the *Alpha Architecture Reference Manual*.

2.5.1 IEEE Floating-Point Conformance

The microprocessor supports the IEEE floating-point operations as defined by the Alpha AXP architecture. Support for a complete implementation of the *IEEE Standard for Binary Floating-Point Arithmetic (754-1985)* is provided by a combination of hardware and software as described in the *Alpha Architecture Reference Manual*.

Additional information that provides guidelines for writing code supporting precise exception handling (necessary for complete conformance to the standard) is in the *Alpha Architecture Reference Manual*. Information specific to the microprocessor follows:

- When operating without the /UNDERFLOW qualifier, the microprocessor replaces underflow results with exact zero, whether the correct result would have been negative zero as defined in the IEEE Standard. This Alpha AXP architecture value-added behavior improves performance over either hardware or software denormal handling.

  When strict IEEE compliance is required, the /UNDERFLOW modifier is necessary and the software must provide the correct result (including negative zero).

- The microprocessor supports infinity operands only when used in the CMPT instruction. Other instructions using infinity operands cause invalid operation (INV) arithmetic traps.

Preliminary

- NaN, denormal, or infinity (except when used in CMPT) input operands produce invalid operation (INV) arithmetic traps when used with arithmetic operation instructions. CPYSE, CPYSN, FCMOV, MF_FPCR, and MT_FPCR are not arithmetic operations, and will pass NaN, denormal, and infinity values without initiating arithmetic traps. Input operand traps take precedence over arithmetic result traps.
- The microprocessor will not produce a NaN, denormal, or infinity result.
- The microprocessor supports IEEE normal and chopped rounding modes in hardware. Instructions designating plus infinity and minus infinity rounding modes cause precise exceptions to the OPCDEC PAL entry point. This implies that the EXC_ADDR register will be loaded with the address of the faulting instruction and all following instructions will be aborted. (Table 3–1 describes the PALcode entry points.)
- The DIVS and DIVT with /INEXACT modifier instructions report an inexact (INE) arithmetic trap on all results of operations that do not involve NaN, infinity, or denormal input operands. Operations using NaN, infinity, and denormal input operands generate invalid operation (INV) arithmetic traps.
- Floating-point exceptions generated by the microprocessor are recorded in two places.
  - Whether or not the corresponding trap is enabled (through the instruction modifiers), the FPCR register (accessible by the MT_FPCR and MF_FPCR instructions) records the occurrence of all detected exceptions (except SWC). Because this register can be cleared only with an explicit clear command (MT_FPCR), the exception information it records is a summary of all exceptions that have occurred since the last clear.
  - If an exception is detected and the corresponding trap is enabled, the microprocessor records the condition in the exception summary register (EXC_SUM, Section 4.2.12) and initiates an arithmetic trap. As a special case, in order to support inexact exception behavior with the DIVS/I and DIVT/I instructions, the FPCR does not record an inexact exception, although the microprocessor always sets the INE bit in the EXC_SUM register during these instructions. This behavior allows software to emulate the division instructions with accurate reporting of potential inexact exceptions.

Preliminary

2.6 Cache Organization

The chip includes two on-chip caches— a data cache and an instruction cache. All memory cells in both caches are fully static, 6-transistor CMOS structures.

2.6.1 Data Cache

The 8KB data cache (Dcache) is a write-through, direct mapped, read-allocate physical cache with 32-byte blocks. When a PCI device writes to cacheable memory, the Dcache block corresponding to the memory address is set invalid.

2.6.2 Instruction Cache

The 8KB instruction cache (Icache) is a physical direct-mapped cache. Each cache block (line), contains the following (also see Figure 9–1):

32 bytes of I-stream data
21-bit associated tag
6-bit address space number (ASN) field
1-bit address space match (ASM) field
8-bit branch history (BHT) field

The Icache does not contain hardware for maintaining coherency with memory and is unaffected by any PCI writes to the memory.

The chip also contains a single-entry Icache stream buffer and supporting logic that reduces the performance penalty incurred by Icache misses during in-line instruction processing. Stream buffer prefetch requests never cross physical page boundaries; instead, they wrap around to the first block of the current page.

2.7 Memory Controller

The on-chip memory controller interfaces the CPU to the system memory and the optional Bcache. It has several memory-mapped control and status registers (CSRs) to program organization, timing, and size of DRAM, VRAM, and Bcache SRAM. It controls CPU requests and DMA requests (from the I/O controller) to and from memory and the Bcache. It also controls VRAM shift register loads and memory refreshes.

The memory controller decodes the address of a CPU request to determine whether the request is for memory or the I/O controller (IOC). It handles the access to the memory controller CSRs, the memory, and Bcache. If the request is directed at the IOC, the memory controller passes control to the IOC.

The memory controller can also perform the following graphics operations:

- Dumb frame buffer operation

Preliminary

- Transparent stipple operation
- Write-per-bit plane masking
- Byte writes (with external gating)
- Full and split VRAM shift-register loads

The memory controller and its CSRs are described in Chapter 5.

2.8 PCI I/O Controller

The on-chip I/O controller (IOC) is the interface between peripheral devices and the CPU and system memory. The IOC interface protocol is complies with the *PCI Local Bus Specification, Revision 2.0*. All peripheral devices in system based on the DECchip 21066 or DECchip 21068 can communicate with the CPU and system memory through the IOC. Peripheral chips using the PCI protocol can be connected directly to the chip without any glue logic. The IOC runs asynchronously to the CPU using the PCI clock input.

The IOC incorporates scatter-gather mapping logic to translate 32-bit addresses generated by PCI bus masters to the 34-bit CPU physical address space. The IOC implements an 8-entry translation look-aside buffer (TLB) for fast translations. Two programmable address windows, controlled by memory-mapped IOC registers, control access from PCI peripherals to system memory.

The IOC and its registers are described in Chapter 6.

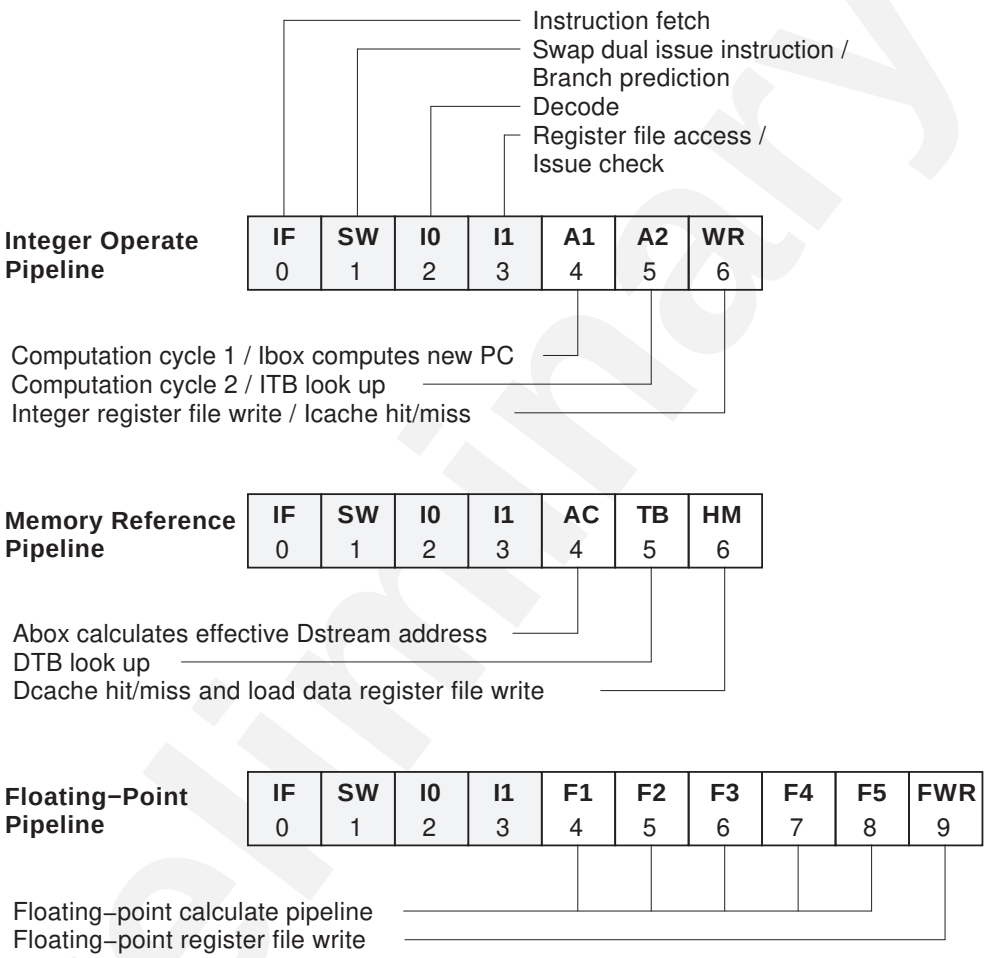
2.9 Pipeline Organization

The microprocessor has a 7-stage pipeline for integer operate and memory reference instructions, and a 10-stage pipeline for floating-point operate instructions. The Ibox maintains state for all pipeline stages, to track outstanding register writes and to determine instruction cache hits and misses.

Figure 2–2 shows the integer operate, memory reference, and floating-point operate pipelines for the Ibox, Ebox, Abox, and FPU. The first four stages of all the pipelines are the same, and are executed in the Ibox. The last stages are unit specific. All of the units have bypassers that allow the results of one instruction to be the operand of a following instruction, without writing the results of the first instruction to a register file.

Preliminary

Figure 2–2 Instruction Pipelines



2.9.1 Static and Dynamic Stages

The integer pipeline divides instruction processing into four static and three dynamic stages of execution. The floating-point pipeline maintains the first four static stages and adds six dynamic stages of execution. The first four stages consist of:

- Instruction fetch
- Swap
- Decode

Preliminary

- Issue logic

These stages are static because instructions can remain valid in the same pipeline stage for multiple cycles while waiting for a resource, or stalling for other reasons.

Dynamic stages always advance state and are unaffected by any stall in the pipeline. (Pipeline stalls are also referred to as pipeline freezes.) A pipeline freeze can occur while no instructions issue, or while one instruction of a pair issues and the second is held at the issue stage. A pipeline freeze implies that a valid instruction or instructions are presented to be issued but cannot proceed.

Upon satisfying all issue requirements, instructions are allowed to continue through any pipeline toward completion. Instructions cannot be held in a given pipe stage after they are issued. It is up to the issue stage to ensure that all resource conflicts are resolved before an instruction is allowed to continue. The only means of stopping instructions after the issue stage is a chip-internal abort condition.

2.9.2 Aborts

There are a number of causes for aborts and they are grouped into two classes:

- Exceptions (including interrupts)
- Nonexceptions

The basic difference between the two classes is that exceptions require the pipeline to be drained of all outstanding instructions before restarting the pipeline at a redirected address. In both classes, all instructions that were fetched after the instruction that caused the abort condition must be flushed from the pipeline. This includes stopping one instruction of a dual-issued pair when an an abort condition occurs on the first instruction of the pair.

The nonexception class does not require draining the pipeline of all outstanding instructions ahead of the aborting instruction. The pipeline can be immediately restarted at a redirected address. Examples of nonexception abort conditions are branch mispredictions, subroutine call or return mispredictions, and instruction cache misses. Dcache misses do not produce abort conditions but can cause pipeline freezes.

If an exception occurs, the processor aborts all instructions issued after the excepting instruction. Due to the nature of some error conditions, this can occur as late as the write cycle. The address of the excepting instruction is then latched in the exception address register (EXC_ADDR, Section 4.2.5). When the pipeline is fully drained, the processor begins to execute instructions at the address given by the PALcode dispatch. The pipeline is drained when:

Preliminary

- All outstanding writes to both the integer and floating-point register files have completed, and arithmetic traps have been reported.
- All outstanding instructions have successfully completed memory management and access protection traps.

2.9.3 Nonissue Conditions

There are two basic reasons for nonissue conditions:

- The pipeline freezes when a valid instruction or pair of instructions are prepared to issue but cannot due to a resource conflict.

  This type of nonissue cycle can be minimized through code scheduling.
- Pipeline *bubbles* when the the pipeline does not contain a valid instruction to issue.

Pipeline bubbles exist due to abort conditions as described in Section 2.9.2. In addition, a single pipeline bubble is produced when a branch-type instruction is predicted to be taken, including subroutine calls and returns. Pipeline bubbles are reduced directly by the hardware through *bubble squashing*, but can also be effectively minimized through careful coding practices. Bubble squashing is the ability of the first four pipeline stages to advance when a bubble is detected in the pipeline stage immediately ahead of it while the pipeline is otherwise frozen.

2.10 Scheduling and Issuing Rules

Sections 2.10.1 through 2.10.5 describe the instruction scheduling and issuing rules.

2.10.1 Instruction Class Definition

This section describes only the performance-related scheduling and dual-issue rules. There are no functional dependencies related to scheduling or dual issue. The scheduling and issuing rules are defined in terms of instruction classes. Table 2–1 specifies all of the instruction classes and the unit that executes the particular class.

Preliminary

Table 2–1 Producer-Consumer Classes

Class	Box	Instruction List
LD	Abox	All loads HW_MFPR, RPCC, RS, RC, STC producers only FETCH consumer only
ST	Abox	All stores and HW_MTPR
IBR	Ebox	Integer conditional branches
FBR	FPU	Floating-point conditional branches
JSR	Ebox	JMP, JSR, RET, JSR_COROUTINE. BSR, BR producer only.
IADDLOG	Ebox	ADDL, ADDL/V, ADDQ, ADDQ/V SUBL, SUBL/V, SUBQ, SUBQ/V, S4ADDL, S4ADDQ, S8ADDL, S8ADDQ, S4SUBL, S4SUBQS8SUBL, S8SUBQ LDA, LDAH AND, BIC, BIS, ORNOT, XOR
SHIFTCM	Ebox	SLL, SRL, SRA EXTQL, EXTLL, EXTWL, EXTBL, EXTQHEXTLH, EXTWH MSKQL, MSKLL, MSKWL, MSKBL, MSKQH, MSKLH, MSKWH INSQL, INSLL, INSWL, INSBL, INSQHINSLH, INSWH ZAP, ZAPNOT CMOVEQ, CMOVNE, CMOVLT, CMOVLE, CMOVGT, CMOVGE, CMOVLBS, CMOVLBC
ICMP	Ebox	CMPEQ, CMPLT, CMPLE, CMPULT, CMPULE, CMPBGE
IMULL	Ebox	MULL, MULL/V
IMULQ	Ebox	MULQ, MULQ/V, UMULH
FPOP	FPU	Floating-point operates except divide
FDIV	FPU	Floating-point divide

2.10.2 Producer-Consumer Latency

The microprocessor enforces the following rules regarding producer-consumer latencies.

The scheduling rules are described as a producer-consumer latency matrix, shown in Figure 2–3. Each row and column in the matrix is a class of Alpha AXP instructions. One cycle of latency is indicated by a 1 in the matrix. One cycle latency means that if instruction B uses the results of instruction A, then instruction B can be issued one cycle after instruction A is issued.

Preliminary

Figure 2–3 Producer-Consumer Latency Matrix

Consumer	Producer									
	LD[1]	JSR	IADDLOG	SHIFTCM	ICMP	IMULL[3]	IMULQ[3]	FPOP	FDIV F/S[4]	FDIV G/T[4]
LD	3	3	2	2	2	21	23	X*	X	X
ST[2]	3	3	2/0	2/0	21/20	21/20	23/22	X/4	X/32	X/61
IBR	3	3	1	2	1	21	23	X	X	X
JSR	3	3	2	2	2	–	–	X	X	X
IADDLOG	3	3	1	2	2	–	–	X	X	X
SHIFTCM	3	3	1	2	2	–	–	X	X	X
ICMP	3	3	1	2	2	–	–	X	X	X
IMUL	3	3	1	2	2	21/19	23/21	X	X	X
FBR	3	X	X	X	X	2	2	6	34	63
FPOP	3	X	X	X	X	2	2	6	34	63
FDIV	3	X	X	X	X	2	2	6	34/30	63/59

1 Dcache hit is assumed for loads. The latency for Dcache miss depends on the system configuration.
2 For some producer classes, two latencies (X/Y) are given with the ST consumer class. X represents the latency for base address of store and Y represents the latency for store data. FLoating–point results cannot be used as the base address for load or store operations.
3 Two latencies are given for IMUL followed by IMUL. The first represents the latency with data dependency; in other words, the second IMUL uses the result from the first IMUL. The second latency is the multiply latency without data dependencies.
4 Two latencies are given for FDIV followed by FDIV. The first represents the latency with data dependency; in other words, the second FDIV uses the result from the first FDIV. The second latency is the division latency without data dependencies.
*X indicatres an impossible state or a state not normally encountered. For example, a floating–point branch would not follow an integer compare.

When determining latency for a given instruction sequence, first identify the classes of each instruction. The following example lists the classes in the comment field:

```
ADDQ    R1, R2, R3              ! IADDLOG class
SRA     R3, R4, R5              ! SHIFT class
SUBQ    R5, R6, R7              ! IADDLOG class
STQ     R7, D(R10)              ! ST class
```

The SRA instruction consumes the result (R3) produced by the ADDQ instruction. The latency associated with an IADD-SHIFT producer-consumer pair as specified by the matrix is one. That means that if the ADDQ was issued in cycle n, the SRA could be issued in cycle n+1.

The SUBQ instruction consumes the result (R5) produced by the SRA instruction. The latency associated with a SHIFT-IADD producer-consumer pair, as specified by the matrix, is two. That means that if the SRA was issued in cycle *n*, the SUBQ could be issued in cycle *n+2*. The Ibox injects one no-operation (NOP) cycle in the pipeline for this case.

Preliminary

In the final case, the STQ instruction consumes the result (R7) produced by the SUBQ instruction. The latency associated with an IADD-ST producer-consumer pair, when the result of the IADD is the store data, is zero. This means that the SUBQ and STQ instruction pair can be dual-issued.

2.10.3 Producer-Producer Latency

Producer-producer latency (also known as write-after-write conflicts) is restricted only by the register write order. For most instructions, this is dictated by issue order; however, IMUL, FDIV, and LD instructions may require more time than other instructions to complete and, therefore, must stall following instructions that write the same destination register to preserve write ordering. In general, only cases involving an intervening producer-consumer conflict are of interest. They can occur commonly in a dual-issue situation when a register is reused. In these cases, producer-consumer latencies are equal to or greater than the required producer-producer latency as determined by write ordering and therefore dictate the overall latency. An example of this case is shown in the code:

```
LDQ R2,D(R0)            ; R2 destination
ADDQ R2,R3,R4           ; wr-rd conflict stalls execution waiting for R2
LDQ R2,D(R1)            ; wr-wr conflict may dual issue when addq issues
```

2.10.4 Instruction Issue Rules

The following conditions prevent instruction issue:

- No instruction can be issued until all of its source and destination registers are clean; in other words, all outstanding writes to the destination register are guaranteed to complete in issue order and either there are no outstanding writes to the source registers or those writes can be bypassed.
- No LD, ST, FETCH, MB, RPCC, RS, RC, TRAPB, HW_MXPR, or BSR BR, JSR (with destination other than R31) instruction can be issued after an MB instruction until the MB has been acknowledged on the external pin bus.
- No IMUL instructions can be issued if the integer multiplier is busy.
- No SHIFT, IADDLOG, ICMP or ICMOV instruction can be issued exactly three cycles before an integer multiplication completes.
- No integer or floating point conditional branch instruction can be issued in the cycle immediately following a JSR, JMP, RET, JSR_COROUTINE, or HW_REI instruction.

Preliminary

- No TRAPB instruction can be issued as the second instruction of a dual-issue pair.
- No LD instructions can be issued in the two cycles immediately following an STC.
- No LD, ST, FETCH, MB, RPCC, RS, RC, TRAPB, HW_MXPR or BSR, BR, JSR (with destination other than R31) instruction can be issued when the Abox is busy due to a load miss or write buffer overflow. For more information see Section 2.4.3.
- No FDIV instruction can be issued if the floating-point divider is busy.
- No floating-point operate instruction can be issued exactly five or exactly six cycles before a floating-point divide completes.

2.10.5 Dual-Issue Table

Table 2–2 can be used to determine instruction pairs that can issue in a single cycle. Instructions are dispatched using two internal data paths or buses. See the *Alpha Architecture Reference Manual* for more information about instructions and their opcodes and definitions.

The buses are referred to in Table 2–2 as IB0, IB1, and IBx.

Any instruction identified with IB0 in the table can be issued in the same cycle as any instruction identified with IB1. An instruction that is identified as IBx may be issued with either IB0 or IB1.

Dual-issue is attempted if the input operands are available as defined by the producer-consumer latency matrix (Figure 2–3) and the following requirements are met:

- Two instructions must be contained within an aligned quadword.
- The instructions must not both be in the group labeled as IB0.
- The instructions must not both be in the group labeled as IB1.
- No more than one of the following instructions can be issued in the same cycle: JSR, integer conditional branch, BSR, HW_REI, BR, or floating-point branch.
- No more than one of the following instructions can be issued in the same cycle: load, store, HW_MTPR, HW_MFPR, MISC, TRAPB, HW_REI, BSR, BR, or JSR.

Preliminary

Producer-consumer latencies of zero indicate that dependent operations between these two instruction classes can dual issue. For example, ADDQ R1, R2, R3 STQ R3, D(R4).

Table 2–2 Opcode Summary with Instruction Issue Bus

	00	08	10	18	20	28	30	38
0/8	PAL IB1	LDA IB0	INTA IB0	MISC IB1	LDF IBx	LDL IBx	BR IB1	BLBC IB1
1/9	Res IB1	LDAH IB0	INTL IB0	HW_MFPR IB1	LDG IBx	LDQ IBx	FBEQ IB0	BEQ IB1
2/A	Res IB1	Res IB1	INTS IB0	JSR IB1	LDS IBx	LDL_L IBx	FBLT IB0	BLT IB1
3/B	Res IB1	LDQ_U IBx	INTM IB0	HW_LD IB1	LDT IBx	LDQ_L IBx	FBLE IB0	BLE IB1
4/C	Res IB1	Res IB1	Res IB1	Res IB1	STF IB0	STL IB1	BSR IB1	BLBS IB1
5/D	Res IB1	Res IB1	FLTV IB1	HW_MTPR IB1	STG IB0	STQ IB1	FBNE IB0	BNE IB1
6/E	Res IB1	Res IB1	FLTI IB1	HW_REI IB1	STS IB0	STL_C IB1	FBGE IB0	BGE IB1
7/F	Res IB1	STQ_U IB1	FLTL IB1	HW_ST IB1	STT IB0	STQ_C IB1	FBGT IB0	BGT IB1

Key to Abbreviations

FLTI — IEEE floating-point instruction opcodes.
FLTL — Floating-point operate instruction opcodes which are data type independent.
FLTV — VAX floating-point instruction opcodes
INTA — Integer arithmetic instruction opcodes
INTL — Integer logical instruction opcodes
INTM — Integer multiply instruction opcodes
INTS — Integer subtract instruction opcodes
JSR — Jump instruction opcodes
MISC — Miscellaneous instruction opcodes
PAL — PALcode instruction (CALL_PAL) opcodes
Res — Reserved for Digital

2.11 PALcode

In a family of machines, both users and operating system implementers require functions to be implemented consistently. When functions conform to a common interface, the code that uses those functions can be used on several different implementations without modification.

Preliminary

Five opcodes are provided by the Alpha AXP architecture as implementation-specific privileged instructions. These instructions are defined independently for each hardware implementation of the Alpha AXP architecture to provide PALcode software routines with access to specific hardware state and functions.

2.11.1 Required PALcode Instructions

The PALcode instructions listed in Table 2–3 are described in the *Alpha Architecture Reference Manual*.

Table 2–3 Required PALcode Instructions

Mnemonic	Type	Operation
HALT	Privileged	Halt processor
IMB	Unprivileged	I-stream memory barrier

2.11.2 Architecturally Reserved PALcode Instructions

The hardware-specific instructions shown in Table 2–4 are executed in the PALcode environment. They produce OPCDEC exceptions if executed while not in the PALcode environment (see Section 3.5 for a definition of OPCDEC). These instructions are mapped using the architecturally reserved opcodes PAL19, PAL1B, PAL1D, PAL1E, PAL1F. They can be used only while executing chip-specific PALcode.

Table 2–4 Implementation-Specific Instructions

Mnemonic	Type	Operation
HW_MTPR	PALmode, Privileged	Move data to processor register
HW_MFPR	PALmode, Privileged	Move data from processor register
HW_LD	PALmode, Privileged	Load data from memory
HW_ST	PALmode, Privileged	Store data in memory
HW_REI	PALmode, Privileged	Return from PALmode exception

Preliminary

Programming Note

PALcode uses the HW_LD and HW_ST instructions to access memory outside of the realm of normal Alpha AXP memory management.

2.12 Physical Address Space

The chip internally uses a 34-bit address space divided as shown in Table 2–5. The most significant two bits of the address <33:32> determine the quadrant of the address space as shown below:

Address <33:32>	Region
00	Memory
01	Graphics memory
	Memory controller and IOC CSRs
	PCI Interrupt ACK/special cycle
	PCI configuration space
10	PCI memory
11	PCI I/O space

The memory region is further divided into cacheable and noncacheable regions. Note that the graphics region, which includes the memory controller and the IOC CSRs, is also noncacheable. Furthermore, PCI peripherals can access only the memory region because the IOC maps the 32-bit PCI address to the internal address that can access only the memory space. The translation mechanism is described in the Chapter 6.

Instructions cannot be executed from the upper three quadrants of memory; that is, when address <33:32> = 01, 10, or 10. Instructions can be executed from the noncacheable region of the lowest quadrant, in which case they will be internally cached in the Icache but not in the Bcache.

When the addresses of CPU-initiated writes are in the graphics region, the memory controller can perform graphics operations as described in Chapter 5. A bank of memory can be mapped to both memory and graphics regions, using the memory controller's bank address mask registers (BMR<3:0>, Section 5.5.2); however, care should taken when using this type of mapping.

Preliminary

Table 2–5 Physical Address Space

Address Region	Address Range <33:0>	Size Gbytes	Cacheable	Graphics Function on Writes
Cacheable Memory	0 0000 0000 0 1FFF FFFF	0.5	Yes	No
Noncacheable Memory	0 2000 0000 0 3FFF FFFF	0.5	No	No
NXM	0 4000 0000 0 FFFF FFFF	3.0	NA*	NA
Graphics memory	1 0000 0000 1 1FFF FFFF	0.5	No	Yes
Memory controller CSRs	1 2000 0000 1 7FFF FFFF	1.5	No	No
IOC CSRs	1 8000 0000 1 BFFF FFFF	0.5	No	No
PCI Interrupt ACK/ Special cycle	1 C000 0000 1 DFFF FFFF	0.25	No	No
PCI configuration	1 E000 0000 1 FFFF FFFF	0.5	No	No
PCI memory	2 0000 0000 2 FFFF FFFF	4.0	No	No
PCI I/O	3 0000 0000 3 FFFF FFFF	4.0	No	No

*NA = Not applicable.

See Appendix B for a list of memory controller and IOC registers and addresses.

2.13 Lock Registers

The chip implements two registers associated with the LDQ_L, LDL_L, STQ_C, and STL_C instructions: the lock_flag register and the locked_physical_address register. The locked range is 32 bytes.

The lock_flag register is cleared on any of the following conditions:

- Execution of any STx_C instruction
- By a DMA write to the locked range
- Execution of a CALL_PAL REI instruction

Preliminary

- When the IOC is locked (see Section 6.3.10)

The lock flag will not be set by any LDx_L instruction while the IOC is locked.

See the *Alpha Architecture Reference Manual* for more information about the lock registers.

Preliminary

3
Privileged Architecture Library Code

3.1 Introduction

This chapter describes the microprocessor's privileged architecture library code (PALcode). The chapter is organized as follows:

- Introduction
- PALcode
- PALmode Environment
- Invoking PALcode
- PALcode Environment Entry Points
- PALcode Restrictions
- PAL Environment Instructions

3.2 PALcode

The Alpha AXP architecture defines an innovative feature called PALcode that allows many different physical implementations to coexist, each one adhering to the same programming interface specification. PALcode has characteristics that make it appear to be a combination of microcode, ROM BIOS, and system service routines, though the analogy to any of these other items is not exact. PALcode exists for several major reasons:

- There are some necessary support functions that are too complex to implement directly in a processor chip's hardware, yet cannot be handled by a normal operating system software routine. Routines to fill the translation buffer, acknowledge interrupts, and dispatch exceptions are some examples. In some architectures, these functions are handled by microcode, but the Alpha AXP architecture is careful not to mandate the use of microcode for reasonable chip implementations.

Preliminary

- There are functions that must run atomically, yet involve long sequences of instructions that may need complete access to all the underlying computer hardware. An example of this is the sequence that returns from an exception or interrupt.
- There are some instructions that are necessary for backward compatibility or ease of programming; however, these are not used often enough to dedicate them to hardware, or are so complex that they would jeopardize the overall performance of the computer. For example, an instruction that does a INTEL-style interlocked memory access might be familiar to someone used to programming on a CISC machine, but is not included in the Alpha AXP architecture. Another example is the emulation of an instruction that has no direct hardware support in a particular chip implementation. In each of these cases, PALcode routines are used to provide the function. The routines are nothing more than programs invoked at specified times, and read in as I-stream code in the same way that all other Alpha AXP code is read. Once invoked, however, PALcode runs in a special mode.

3.3 PALmode Environment

PALcode runs in a special environment called PALmode, defined as follows:

- I-stream memory mapping is disabled. Because the PALcode is used to implement translation buffer fill routines, I-stream mapping clearly cannot be enabled.
- The program has privileged access to all the computer hardware. Most of the functions handled by PALcode are privileged and need control of the lowest levels of the system.
- Interrupts are disabled. If a long sequence of instructions need to be executed atomically, interrupts cannot be allowed.

One important aspect of PALcode is that it uses normal Alpha AXP instructions for most of its operations; that is, the same instruction set that nonprivileged Alpha AXP programmers use. There are a few extra instructions that are only available in PALmode, and will cause an OPCDEC exception if attempted while not in PALmode. The Alpha AXP architecture allows some flexibility in what these special PALmode instructions do. In the microprocessor the special PALmode-only instructions perform the following functions:

- Read or write internal processor registers (HW_MFPR, HW_MTPR)
- Perform memory load or store operations without invoking the normal memory management routines (HW_LD, HW_ST)

Preliminary

- Return from an exception or interrupt (HW_REI)

Refer to Section 3.7 for detail information on these special PALmode instructions.

When executing in PALmode, there are certain restrictions for using the privileged instructions because PALmode gives the programmer complete access to many of the internal details of the microprocessor.

Caution

It is possible to cause unintended side effects by writing what appears to be perfectly acceptable PALcode. As such, PALcode is not something that many users will want to change.

Refer to Section 3.6 for additional information on PALmode restrictions.

3.4 Invoking PALcode

PALcode is invoked at specific entry points, under certain well-defined conditions. PALcode can be thought of as a series of callable routines, with each routine indexed as an offset from a base address. The base address of the PALcode is programmable (stored in the PAL_BASE internal processor register), and is normally set by the system reset code. Refer to Section 3.5 for additional information on PALcode entry points.

When an event occurs that needs to invoke PALcode, the microprocessor first drains the pipeline. The current PC is loaded into the EXC_ADDR internal processor register, and the appropriate PALcode routine is dispatched also. These operations occur under direct control of the chip hardware, and the machine is now in PALmode. When the HW_REI instruction is executed at the end of the PALcode routine, the hardware executes a jump to the address contained in the EX_ADDR internal processor register. The LSB is used to indicate PALmode to the hardware. Generally, upon return from a PALcode routine, the LSB is clear, in which case the hardware will load the new PC, enables interrupts, enables memory mapping, and dispatches back to the user.

The most basic use of PALcode is to handle complex hardware events, and it is called automatically when the particular hardware event is sensed. This use of PALcode is similar to other architectures' use of microcode. There are several major categories of hardware-initiated PALcode:

- When the microprocessor is reset, it enters PALmode and executes the RESET PALcode. The system will remain in PALmode until a HW_REI instruction is executed and EXC_ADDR bit 0 is cleared. It then continues

Preliminary

execution in nonPALmode (native mode), as just described. It is during this initial RESET PALcode execution that the rest of the low level system initialization is performed, including any modification to the PALcode base register.

- When a system hardware error is detected by the microprocessor, it invokes one of several PALcode routines, depending upon the type of error. Errors such as machine checks, arithmetic exceptions, reserved or privileged instruction decode, and data fetch errors are handled in this manner.
- When the microprocessor senses an interrupt, it dispatches the acknowledgment of the interrupt to a PALcode routine that does the necessary information gathering, then handles the situation appropriately for the given interrupt.
- When a D-stream or I-stream translation buffer miss occurs, one of several PALcode routines is called to perform the TB fill. The memory management algorithms or even the existence of a virtual to physical page mapping is flexible. In the simplest case, this could be an automatic one-to-one translation from virtual to physical address. On a normal operating system these routines would consult page tables and perform the translation and fill based upon its contents.

These elements are all very basic hardware-related functions, and would be difficult to efficiently implement using normal operating system service routines.

3.4.1 CALL_PAL Instruction

The other mechanism used to invoke PALcode is the CALL_PAL. This is a special instruction that dispatches to PALcode at a specific entry point using the same set of steps as the hardware-activated PALcode. That is, the pipeline is drained, the PC is saved, and the appropriate dispatch to an offset into the PALcode base is performed. The only difference is that the dispatch is controlled by the program through an instruction, rather than through a hardware event or error. Also, PAL_CALLs place PC+4 in the EXC_ADDR. The CALL_PAL instruction format includes a single parameter, the function field, that defines which CALL_PAL routine to invoke. Only a subset of all the possible CALL_PAL function values are supported with hardware dispatches, in the DECchip 21066 and DECchip 21068. These dispatches are described in Section 3.5. CALL_PAL routines can perform different functions for different operating systems running on the microprocessor. Unlike the basic hardware-generated PALcode, the CALL_PAL operations are largely optional and based upon what the system implementation needs.

Preliminary

There is a subtle difference between the two basic uses of PALcode: hardware-dispatched and CALL_PAL-dispatched. The hardware-invoked PALcode functions are necessary in some form for almost any useful computer system. For example, when the microprocessor detects a serious system error, it will dispatch to the machine check (MCHK) PALcode entry point. The exact PALcode that resides at this entry point can do whatever is reasonable, based upon system needs.

The CALL_PAL instruction is totally under the control of the executing program for dispatch. If the program never executes one of the instructions that is included in the CALL_PAL list, then none of that PALcode will ever be run. Even here, the PALcode that does run once invoked is executing in PALmode and is under the same restrictions as the hardware-activated PALcode.

The microprocessor supports hardware dispatch for both privileged and nonprivileged CALL_PAL instructions. That is, some of the functions that are passed to the CALL_PAL instruction are considered special. The designation of privileged or nonprivileged refers to whether the user can call that particular CALL_PAL, and not the mode that it eventually runs in. Without exception, every CALL_PAL in struction will dispatch to PALcode that runs in PALmode. Only kernel users can call privileged CALL_PAL instructions.

The difference between privileged and nonprivileged CALL_PAL instructions is that the former can only be executed in kernel mode. Otherwise, they are vectored to 13E0 (OPCDEC).

These are both CALL_PAL instructions, dispatched in exactly the same way, and when executed enter PALmode, do their function, and return to the user. The only difference is that, before execution, a check is made to determine if the user is in the correct mode. If a nonkernel mode user attempts to execute a privileged CALL_PAL instruction, an OPCDEC PALcode routine is run instead of the CALL_PAL function. In addition, if a CALL_PAL function code that is not supported by the microprocessor hardware dispatch is attempted, an OPCDEC exception is taken.

3.5 PALcode Entry Points

Table 3–1 prioritizes entry points from highest to lowest priority; the first row in the table (reset) has the highest priority. The table defines only the entry point offset, bits <13:0>. The high-order bits of the new PC (bits <33:14>) come from the PAL_BASE register. The PAL_BASE register value at powerup is equal to zero.

Preliminary

Note

PALcode at PAL entry points of higher priority than DTB_MISS must unlock possible MM_CSR register and VA register locks.

Table 3–1 PALcode Entry Points

Entry Name	Time	Offset	Cause
RESET	Anytime	0000	
MCHK	Anytime	0020	Uncorrected hardware error
ARITH	Anytime	0060	Arithmetic exception
INTERRUPT	Anytime	00E0	Includes corrected hardware error
D-stream Errors	Pipe_stage 6	01E0, 08E0, 09E0, 11E0	See Table 3–2.
ITB_MISS	Pipe stage 5	03E0	ITB miss
ITB_ACV	Pipe stage 5	07E0	I-stream access violation
CALL_PAL	Pipe stage 5	2000, 2040, 2080 20C0 through 3FC0	128 locations based on instruction 7,5..0. See the next table entry.
OPCDEC	Pipe stage 5	13E0	Reserved or privileged opcode
FEN	Pipe stage 5	17E0	FP operation attempted with: • FP instructions disabled by ICCSR FPE bit • FP IEEE round to ± infinity • FP IEEE with data type field other than S, T, Q

PALcode functions are implemented by way of the CALL_PAL instruction. CALL_PAL instructions cause exceptions in the hardware. As with all exceptions, the EXC_ADDR register is loaded by hardware with a possible return address.

CALL_PAL exceptions do not load the EXC_ADDR register with the address of the CALL_PAL instruction. They load the EXC_ADDR register with the address of the instruction following the CALL_PAL. PALcode supporting the desired PALmode function need not increment the EXC_ADDR register before executing a HW_REI instruction to return to native mode. This feature requires special handling in the arithmetic trap and machine check PALcode flows. See Section 4.2.5 for more information.

To improve speed of execution, a limited number of CALL_PAL instructions are directly supported in hardware with dispatches to specific address offsets.

Preliminary

The microprocessor provides the first 64 privileged and 64 unprivileged CALL_PAL instructions with regions of 64 bytes. This produces hardware PAL entry points described as follows:

- Privileged CALL_PAL Instructions <00000000:0000003F>
  Offset = 2000_{16} + (<5:0> shift left 6)
- Unprivileged CALL_PAL Instructions <00000080:000000BF>
  Offset = 3000_{16} + (<5:0> shift left 6)

The CALL_PAL instructions that do not fall within the ranges <00000000:0000003F> and <00000080:000000BF> result in an OPCDEC exception. CALL_PAL instructions that fall within the range <00000000:0000003F> while the microprocessor is not executing in kernel mode will result in an OPCDEC exception.

The hardware recognizes four classes of D-stream memory management errors.

- Bad virtual address (incorrect sign extension)
- DTB miss
- Alignment error
- ACV, FOR, FOW

The following errors get mapped into four PAL entry points:

- UNALIGN
- DTB_MISS PAL Mode
- DTB_MISS Native Mode
- D_FAULT

Table 3–2 lists the priority of these entry points as a group with respect to each of the other entry points. A particular D-stream memory reference may generate errors that fall into more than one of the four error classes which the hardware recognizes.

Table 3–2 D-stream Error PAL Entry Points

BAD_VA	DTB_MISS	UNALIGN	PAL	Other	Offset
1	x	1	x	x	11E0 UNALIGN
1	x	0	x	x	01E0 D_FAULT

(continued on next page)

Preliminary

Table 3–2 (Cont.) D-stream Error PAL Entry Points

BAD_VA	DTB_MISS	UNALIGN	PAL	Other	Offset
0	1	x	1	x	09E0 DTB_MISS PAL
0	1	x	0	x	08E0 DTB_MISS NATIVE
0	0	1	x	x	11E0 UNALIGN
0	0	0	x	1	01E0 D_FAULT

See Section 3.4 for additional information on PALcode entry points.

3.6 PALmode Restrictions

Many of the PALmode restrictions involve waiting n cycles before using the results of a PAL instruction. Inserting n instructions between the two time-sensitive instructions is the typical method of waiting for n cycles. Because the microprocessor can dual issue instructions it is possible to write code that requires 2*n+1 instructions to wait n cycles. Due to the resource requirements of individual instructions and the microprocessor's hardware design, multiple copies of the same instruction cannot be dual issued. This is used in some of the following examples. The following is a list of PALmode restrictions:

- As a general rule, HW_MTPR instructions require at least 4 cycles to update the selected IPR. At least three cycles of delay must be inserted before using the result of the register update.

  The following instructions will pipeline correctly and do not require software timing except for accesses of the TB registers.

  – Multiple reads

  – Multiple writes

  – Read followed by write

  These cycles can be guaranteed by either including seven instructions, which do not use the IPR in transition, or proving through the dual issue rules and/or state of the machine, that at least three cycles of delay will occur. Multiple copies of a HW_MTPR instruction (used as a NOP instruction) can be used to pad cycles after the original HW_MTPR. Multiple copies of the same instruction will never dual issue. Because of this, the maximum number of instructions necessary to insure at least three cycles of delay is three.

Preliminary

Example:

```
HW_MTPR Rx, HIER        ; Write to HIER
HW_MFPR R31, 0          ; NOP mxpr instruction
HW_MFPR R31, 0          ; NOP mxpr instruction
HW_MFPR R31, 0          ; NOP mxpr instruction
HW_MFPR Ry, HIER        ; Read from HIER
```

The HW_REI instruction uses the ITB if the EXC_ADDR register contains a nonPALmode virtual program counter (VPC bit 0 = 0). By the previous rule, it is implied that at least three cycles of delay must be included after writing the ITB before executing a HW_REI instruction to exit PALmode.

Exceptions:

– HW_MFPR instructions reading a PAL_TEMP register can never occur exactly two cycles after a HW_MTPR instruction writing a PAL_TEMP register. The solution results in code as follows:

```
HW_MTPR Rx, PAL_R0      ; Write PAL temp [0]
HW_MFPR R31, 0          ; NOP mxpr instruction
HW_MFPR R31, 0          ; NOP mxpr instruction
HW_MFPR R31, 0          ; NOP mxpr instruction
HW_MFPR Ry, PAL_R0      ; Read PAL temp [0]
```

  This code guarantees three cycles of delay after the write before the read. It is also possible to make use of the cycle immediately following a HW_MTPR instruction to execute a HW_MFPR instruction to the same (accomplishing a swap) or a different PAL_TEMP register. The swap operation only occurs if the HW_MFPR instruction immediately follows the HW_MTPR. This timing requires great care and knowledge of the pipeline to ensure that the second instruction does not stall for one or more cycles. Use of the slot to accomplish a read from a different PAL_TEMP register requires that the second instruction will not stall for exactly one cycle. This is much easier to insure. A HW_MFPR instruction can stall for a single cycle as a result of a write-after-write conflict.

– The EXC_ADDR register can be read by a HW_REI instruction only two cycles after the HW_MTPR. This is equivalent to one intervening cycle of delay. This translates to code as follows:

```
HW_MTPR Rx, EXC_ADDR    ; Write EXC_ADDR
HW_MFPR R31, 0          ; NOP cannot dual issue with
HW_REI                  ; either Return
```

Preliminary

- An HW_MTPR operation to the DTBIS register cannot be sourced from a bypassed path. All data being moved to the DTBIS register must be sourced directly from the register file. One way to ensure this is to provide at least 3 cycles of delay before using the result of any integer operation (except MUL) as the source of an HW_MTPR DTBIS.

  Note

  Do not use a MUL as the source of DTBIS data.

  Code for this operation is:

```
ADDQ R1,R2,R3           ; source for DTBIS address
ADDQ R31,R31,R31        ; cannot dual issue with above, 1st cycle of delay
ADDQ R31,R31,R31        ; 2nd cycle of delay
ADDQ R31,R31,R31        ; 3rd cycle of delay
ADDQ R31,R31,R31        ; may dual issue with below, else 4th cycle of delay
HW_MTPR R3,DTBIS        ; R3 must be in register file, no bypass possible
```

- At least one cycle of delay must occur after a HW_MTPR TB_CTL before a HW_MTPR ITB_PTE or a HW_MFPR ITB_PTE. This must be done to allow setup of the ITB large page or small page decode.
- The first cycle (the first one or two instructions) at all PALcode entry points can not execute a conditional branch instruction or any other instruction that uses the JSR stack hardware. This includes instructions:

  JSR
  JMP
  RET
  JSR_COROUTINE
  BSR
  HW_REI
  All Bxx opcodes except BR

- Table 3–3 lists the number of cycles required after a HW_MTPR instruction before a subsequent HW_REI instruction for the specified IPRs. These cycles can be ensured by inserting one HW_MFPR R31,0 instruction or other appropriate instruction(s) for each cycle of delay required after the HW_MTPR.

Preliminary

Table 3–3 HW_MTPR Restrictions

IPR	Cycles Between HW_MTPR and HW_REI
DTBIS, ASM, ZAP	0
ITBIS, ASM, ZAP	2
HIER, SIER	3
HIRR, SIRR	3
ICCSR [FPE]	4
PS	4

- When loading the CC register, bits <3:0> must be loaded with zero. Loading nonzero values in these bits can cause an inaccurate count.
- An HW_MTPR DTBIS cannot be combined with an HW_MTPR ITBIS instruction. The hardware will not clear the ITB if both the Ibox and Abox IPRs are simultaneously selected. Two instructions are needed to clear each TB individually. Code for this operation is:

```
HW_MTPR Rx,ITBIS
HW_MTPR Ry,DTBIS
```

- Three cycles of delay are required between:

HW_MTPR HIER, SIER	and	HW_MFPR HIRR, SIRR
HW_MTPR HIRR, SIRR	and	HW_MFPR xIRR
MTPS	and	LD or ST
MTPS	and	HW_MFPR xIRR
HW_MTPR ALT_MODE	and	HW_LD/HW_ST ALT_MODE

- The following operations are disabled in the cycle immediately following a HW_REI instruction:
  - HW_MxPR ITB_TAG
  - HW_MxPR ITB_PTE
  - HW_MxPR ITB_PTE_TEMP

    This rule implies that it is not a good idea to ever allow exceptions while updating the ITB. The ITB register will not be written if:

    - An exception interrupts flow of the ITB miss routine and attempts to REI back. The return address begins with a HW_MxPR instruction to an ITB register.

Preliminary

- The REI is predicted correctly to avoid any delay between the two instructions.

The code for this operation is:

```
HW_REI                  ; return from interrupt
HW_MTPR R1,ITB_TAG      ; attempts to execute very next cycle, instr ignored
```

- The following registers can only be accessed in PALmode:
  - ITB_TAG
  - ITB_PTE
  - ITB_PTE_TEMP

  If the instruction HW_MTPR or HW_MFPR is to or from the previous mentioned registers while not in PALmode by setting the HWE (hardware enable) bit of the ICCSR, the instructions will be ignored.
- When writing the PAL_BASE register, exceptions can not occur. An exception occurring simultaneously with a write to the PAL_BASE can leave the register in a metastable state. All asynchronous exceptions but reset can be avoided under the following conditions:
  - PALmode: Blocks all interrupts
  - Machine checks disabled: Blocks I/O error exceptions (by way of the ABOX_CTL reg or MB isolation)
  - Not under trap shadow: Avoids arithmetic traps

  The trap shadow is defined as:

  Less than 3 cycles after a nonmul integer operate that may overflow
  Less than 22 cycles after a MULL/V instruction
  Less than 24 cycles after a MULQ/V instruction
  Less than 6 cycles after a nondiv fp operation that may cause a trap
  Less than 34 cycles after a DIVF or DIVS that may cause a trap
  Less than 63 cycles after a DIVG or DIVT that may cause a trap
- The sequence HW_MTPR PTE, HW_MTPR TAG is not allowed. At least two null cycles must occur between HW_MTPR PTE and HW_MTPR TAG.
- The MCHK exception service routine must check the EXC_SUM register for simultaneous arithmetic errors. Arithmetic traps will not trigger exceptions a second time after returning from exception service for the machine check.

Preliminary

- Three cycles of delay must be inserted between HW_MFPR DTB_PTE and HW_MFPR DTB_PTE_TEMP. Code for this operation is:

```
HW_MFPR Rx,DTB_PTE              ; reads DTB_PTE into DTB_PTE_TEMP register
HW_MFPR R31,0                   ; 1st cycle of delay
HW_MFPR R31,0                   ; 2nd cycle of delay
HW_MFPR R31,0                   ; 3rd cycle of delay
HW_MFPR Ry,DTB_PTE_TEMP         ; read DTB_PTE_TEMP into register file Ry
```

- Three cycles of delay must be inserted between HW_MFPR ITB_PTE and HW_MFPR ITB_PTE_TEMP. Code for this operation is:

```
HW_MFPR Rx,DTB_PTE              ; reads DTB_PTE into DTB_PTE_TEMP register
HW_MFPR R31,0                   ; 1st cycle of delay
HW_MFPR R31,0                   ; 2nd cycle of delay
HW_MFPR R31,0                   ; 3rd cycle of delay
HW_MFPR Ry,DTB_PTE_TEMP         ; read DTB_PTE_TEMP into register file Ry
```

- The content of the destination register for HW_MFPR Rx,DTB_PTE or HW_MFPR Rx,ITB_PTE is UNPREDICTABLE.
- Two HW_MFPR DTB_PTE instructions cannot be issued in consecutive cycles. This implies that more than one instruction can be necessary between the HW_MFPR instructions if dual issue is possible. Similar restrictions apply to the ITB_PTE register.
- Reading the EXC_SUM register requires special timing. Refer to Section 4.2.12 for specific information.
- DMM errors occurring one cycle before HW_MxPR instructions to the ITB_PTE will not stop the TB pointer from incrementing to the next TB entry even though the HW_MxPR instruction will be aborted by the DMM error. This restriction only affects performance and not functionality.
- PALcode that writes multiple ITB entries must write the entry that maps the address contained in the EXC_ADDR register last.
- HW_ST instructions cannot be followed, for two cycles, by any load instruction that may miss in the Dcache.
- Updates to the ASN field of the ICCSR IPR require at least 10 cycles of delay before entering native mode that can reference the ASN during Icache access. If the ASN field is updated in Kernel mode by way of the HWE bit of the ICCSR IPR, it is sufficient that all I-stream references during this time be made to pages with the ASM bit set to avoid use of the ASN.
- HW_MTPR instructions that update the TB_CTL register cannot follow a HW_MTPR instruction that updates the DTB_PTE or ITB_PTE register by one cycle.

Preliminary

- The HW_MTPR instructions that update the following IPRs require delays as shown in Table 3–4:
  - ICCSR (ASN field)
  - FLUSH_IC
  - FLUSH_IC_ASM

  The purpose of the delay is to ensure that the update occurs before the first instruction fetch in native mode, since the pipeline may currently contain instructions that were fetched before the update (which would remain valid during a pipeline stall). It is necessary that at least one instruction is issued during each cycle of the delay to ensure that the pipeline is cleared of all instructions fetched prior to the update.

  If the update is performed in kernel mode through the use of the HWE bit of the ICCSR, it is sufficient that all I-stream references during this time be made to pages with the ASM bit set to avoid use of the ASN.

Table 3–4 HW_MTPR Cycle Delay

IPR	Cycles Delay
ICCSR (ASN field only)	8
FLUSH_IC	9
FLUSH_IC_ASM	9

- Machine check exceptions taken while in PALmode can load the EXC_ ADDR register with a restart address one instruction earlier than the correct restart address. Some HW_MxPR instructions may have already completed execution even if the restart address indicates the HW_MxPR as the return instruction. Re-execution of some HW_MxPR instructions can alter the machine state (that is, TB pointers, EXC_ADDR register mask).

  The mechanism used to stop instruction flow during machine check exceptions causes the machine check exception to appear as a D-stream fault on the following instruction in the hardware pipeline. In the event that the following instruction is a HW_MxPR, a D-stream fault will not abort execution in all cases. The EXC_ADDR will be loaded with the address of the HW_MxPR instruction as if it were aborted. A HW_REI to this restart address will incorrectly re-execute this instruction.

  Machine check service routines should check for MxPR instructions at the return address before continuing.

Preliminary

3.6.1 Translation Buffer Miss Flows

This section describes hardware specific details to aid the PALcode programmer in writing ITB and DTB fill routines. These flows highlight the tradeoffs and restrictions between PAL and hardware. The PALcode source that is released with the chip should be consulted before any new flows are written. A working knowledge of the Alpha AXP memory management architecture is assumed.

3.6.1.1 Instruction Translation Buffer Miss

When the Ibox encounters an ITB miss it:

1. Latches the VPC of the target instruction-stream reference in the EXC_ ADDR IPR.
2. Flushes the pipeline of any instructions following the instruction which caused the ITB miss.
3. Waits for any other instructions which may be in progress to complete.
4. Enters PALmode.
5. Jumps to the ITB miss PAL entry point.

The recommended PALcode sequence for translating the address and filling the ITB is as follows:

1. Create some scratch area in the integer register file by writing the content of a few integer registers to the PAL_TEMP register file.
2. Read the target virtual address from the EXC_ADDR.
3. Fetch the PTE (this can take multiple reads) using a physical-mode HW_ LD instruction. If this PTE's valid bit is clear, report TNV or ACV as appropriate.
4. The Alpha Architecture Handbook states that translation buffers cannot contain invalid PTEs; the PTE's valid bit must be explicitly checked by PALcode. Since the ITB's PTE RAM does not hold the FOE bit, the PALcode must also explicitly check this condition. If the PTE's valid bit is set and FOE bit is clear, PALcode can fill an ITB entry.
5. Write the original virtual address to the TB_TAG register using HW_ MTPR. This writes the TAG into a temp register and not the actual tag field in the ITB.
6. Write the PTE to the TB_CTL to select between the large page or small page TB regions. Wait at least one cycle before executing the next step.
7. Write the PTE to the ITB_PTE register using HW_MTPR. This HW_MTPR causes both the TAG and PTE fields in the ITB to be written.

Preliminary

8. Restore the contents of any modified integer registers to their original state using the HW_MFPR instruction.
9. Restart the instruction stream using the HW_REI instruction.

Note

It is not necessary to delay issuing the HW_MTPR to the ITB_PTE after the MTPR to the ITB_TAG is issued.

3.6.1.2 Data Translation Buffer Miss

When the Abox encounters a DTB miss it:

1. Latches the referenced virtual address in the VA IPR and other information about the reference in the MM_CSR IPR.
2. Locks the VA and MM_CSR registers against further modifications. Latches the PC of the instruction that generated the reference in the EXC_ ADDR register.
3. Drains the machine as described in Section 3.6.1.1.
4. Jumps to the DTB miss PALcode entry point.

Unlike ITB misses, DTB misses can occur while the CPU is executing in PALmode. The recommended PALcode sequence for translating the address and filling the DTB is as follows:

1. Create some scratch area in the integer register file by writing the contents of a few integer registers to the PAL_TEMP register file.
2. Read the requested virtual address from the VA register. The act of reading this register unlocks the VA and MM_CSR registers. The MM_CSR register is updated only when D-stream memory management errors occur. It will retain information about the instruction which generated the DTB miss. This can be useful later.
3. Fetch the Page Table Entry (PTE). This operation can require multiple reads. If the Valid bit of the PTE is clear, a Translation Not Valid (TNV) or Access Violation (ACV) must be reported unless the instruction which caused the DTB miss was FETCH or FETCH_M. This can be checked by way of the opcode field of the MM_CSR register. If the value in this field is 18 (hex), then a FETCH or FETCH_M instruction caused this DTB miss. As mandated in the Alpha Architecture Handbook, the subsequent TNV or ACV should not be reported. Therefore PALcode should:
   a. Read the value in EXC_ADDR IPR

Preliminary

b. Increment the value by four

c. Write the value back to EXC_ADDR IPR

d. Execute a HW_REI.

4. Write the register which holds the contents of the PTE to the DTB_CTL. This has the effect of selecting one of the four possible granularity hint sizes.
5. Write the original virtual address to the TB_TAG register. This writes the TAG into a temp register and not the actual tag field in the DTB.
6. Write the PTE to the DTB_PTE register. This HW_MTPR causes both the TAG and PTE fields in the DTB to be written.

Note

It is not necessary to delay issuing the HW_MTPR to the DTB_PTE after the MTPR to the DTB_TAG is issued.

7. Restore the contents of any modified integer registers.
8. Restart the instruction stream using the HW_REI instruction.

3.7 Implementation of the Architecturally Reserved Instructions

PALcode uses the Alpha AXP instruction set for most of its operations. The microprocessor maps the architecturally reserved opcodes (PALRES0 - PALRES4) to:

- A special load and store (HW_LD, HW_ST)
- A move-to and a move-from processor register (HW_MTPR, HW_MFPR)
- A return from PALmode exception or interrupt (HW_REI)

These instructions produce an OPCDEC exception if executed while not in the PALmode environment. If the HWE bit of the ICCSR internal processor register (IPR) is set, these instructions can be executed in kernel mode.

Register checking and bypassing logic is provided for PALcode instructions as it is for nonPALcode instructions, when using general purpose registers.

Preliminary

Note

Explicit software timing is required for accessing the hardware specific internal processor registers (IPR) and the PAL_TEMPs. These constraints are described in the PALmode restriction and IPR sections.

3.7.1 HW_MFPR and HW_MTPR Instructions

The internal processor register (IPR) specified by the PAL, ABX, IBX, and index field is written/read with the data from the specified integer register.

Caution

Internal processor registers can have side effects that happen as the result of writing/reading them.

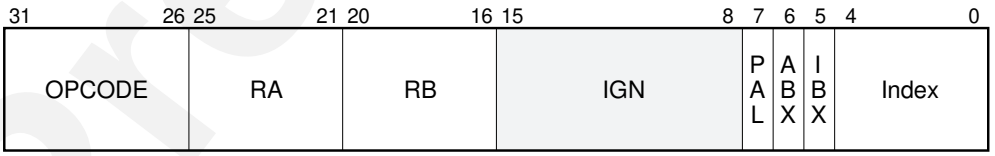
Coding restrictions (see Section 3.6) are associated with accessing various registers. Separate bits are used to access the:

- Abox IPRs
- Ibox IPRs
- PAL_TEMPs

It is possible for an HW_MTPR instruction to write multiple registers in parallel if they both have the same index.

Figure 3–1 shows the HW_MFPR and HW_MTPR instruction format. Table 3–5 lists the instructions fields.

Figure 3–1 HW_MFPR and HW_MTPR Instruction Format



Preliminary

Table 3–5 HW_MFPR and HW_MTPR Format Description

Bits	Field	Description
31:26	OPCODE	25 (HW_MFPR) or 29 (HW_MTPR)
25:16	RA/RB	Contains the source (HW_MTPR) or destination (HW_MFPR) register number. The RA and RB fields must always be identical.
7	PAL	If set, this HW_MFPR or HW_MTPR instruction is referencing a PAL temporary register, PAL_TEMP.
6	ABX	If set, this HW_MFPR or HW_MTPR instruction is referencing a register in the Abox.
5	IBX	If set, this HW_MFPR or HW_MTPR instruction is referencing a register in the Ibox.
4:0	INDEX	Index within the group

Table 3–6 indicates how the PAL, ABX, IBX, and INDEX fields are set to access the IPRs. Setting the PAL, ABX, and IBX fields to zero generates a NOP.

Table 3–6 Internal Processor Register Access

Mnemonic	PAL	ABX	IBX	INDEX	Access	Comments
TB_TAG	x	x	1	0	W	PALmode only
ITB_PTE	x	x	1	1	R/W	PALmode only
ICCSR	x	x	1	2	R/W	
ITB_PTE_TEMP	x	x	1	3	R	PALmode only
EXC_ADDR	x	x	1	4	R/W	
SL_RCV	x	x	1	5	R	
ITBZAP	x	x	1	6	W	PALmode only
ITBASM	x	x	1	7	W	PALmode only
ITBIS	x	x	1	8	W	PALmode only
PS	x	x	1	9	R/W	
EXC_SUM	x	x	1	10	R/W	
PAL_BASE	x	x	1	11	R/W	
HIRR	x	x	1	12	R	
SIRR	x	x	1	13	R/W	

(continued on next page)

Preliminary

Table 3–6 (Cont.) Internal Processor Register Access

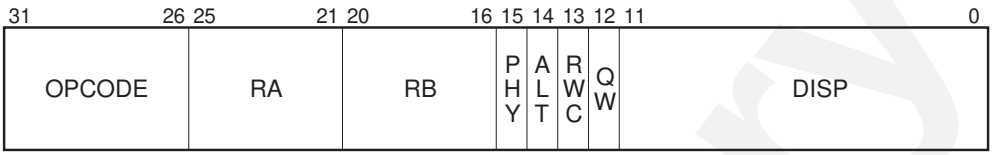
Mnemonic	PAL	ABX	IBX	INDEX	Access	Comments
ASTRR	x	x	1	14	R/W	
HIER	x	x	1	16	R/W	
SIER	x	x	1	17	R/W	
ASTER	x	x	1	18	R/W	
SL_CLR	x	x	1	19	W	
SL_XMIT	x	x	1	22	W	
DTB_CTL	x	1	x	0	W	
DTB_PTE	x	1	x	2	R/W	
DTB_PTE_TEMP	x	1	x	3	R	
MM_CSR	x	1	x	4	R	
VA	x	1	x	5	R	
DTBZAP	x	1	x	6	W	
DTBASM	x	1	x	7	W	
DTBIS	x	1	x	8	W	
DC_STAT	x	1	x	12	R	
ABOX_CTL	x	1	x	14	W	
ALT_MODE	x	1	x	15	W	
CC	x	1	x	16	W	
CC_CTL	x	1	x	17	W	
FLUSH_IC	x	1	x	21	W	
FLUSH_IC_ASM	x	1	x	23	W	
PAL_TEMP<31:0>	1	x	x	31..0	R/W	

3.7.2 HW_LD and HW_ST Instructions

PALcode uses the HW_LD and HW_ST instructions to access memory outside of the realm of normal Alpha AXP memory management. Figure 3–2 shows the HW_LD and HW_ST instructions format. Table 3–7 lists the instruction fields.

Preliminary

Figure 3–2 HW_LD and HW_ST Instructions Format



The effective address of these instructions is calculated as follows:

addr $\Leftarrow$ (SEXT(DISP) + RB) AND NOT (QW | 11_2)

Table 3–7 HW_LD and HW_ST Format Description

Bits	Field	Description
31:26	OPCODE	Is either 27 (HW_LD) or 31 (HW_ST).
25:16	RA/RB	Contains register numbers, interpreted in the normal fashion for loads and stores.
15	PHY	If clear, the effective address of the HW_LD or HW_ST is a virtual address. If set, then the effective address of the HW_LD or HW_ST is a physical address.
14	ALT	For virtual-mode HW_LD and HW_ST instructions, this bit selects the processor mode bits that are used for memory management checks. If ALT is clear, the current mode bits of the PS register are used; if ALT is set the mode bits in the ALT_MODE IPR are used. Physical-mode load-lock and store-conditional variants of the HW_LD and HW_ST instructions may be created by setting both the PHY and ALT bits.
13	RWC	The RWC (read-with-write check) bit, if set, enables both read and write access checks on virtual HW_LD instructions.
12	QW	The quadword bit specifies the data length. If it is set then the length is quadword. If it is clear then the length is longword.
11:0	DISP	The DISP field holds a 12-bit signed byte displacement.

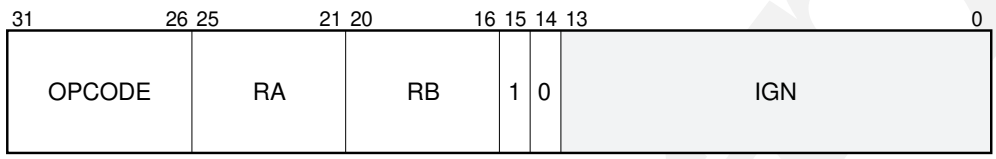
3.7.3 HW_REI Instruction

The HW_REI instruction uses the address in the Ibox EXC_ADDR register to determine the new virtual program counter (VPC). Bit 0 of the EXC_ADDR indicates the state of the PALmode bit on the completion of the HW_REI. If EXC_ADDR bit 0 is set, then the processor remains in PALmode. This allows PALcode to transition from PALmode to nonPALmode. The HW_REI instruction can also be used to jump from PALmode to PALmode. This allows PAL instruction flows to take advantage of the D-stream mapping hardware in

Preliminary

the chip, including traps. Figure 3–3 shows the HW_REI instruction format. Table 3–8 lists the instruction fields.

Figure 3–3 HW_REI Instruction Format



Bits <15:14> contain the branch prediction hint bits. The microprocessor pushes the contents of the EXC_ADDR register on the JSR prediction stack. Bit 15 must be set to pop the stack to avoid misalignment.

The next address and PALmode bit are calculated as follows:

VPC ⇐ EXC_ADDR AND {NOT 3}
PALmode ⇐ EXC_ADDR[0]

Table 3–8 HW_REI Format Description

Bits	Field	Description
31:26	OPCODE	The OPCODE field contains 30.
25:16	RA/RB	Contains register numbers which should be R31 or a stall may occur.

Preliminary

4
Internal Processor Registers

4.1 Introduction

This chapter describes the internal processor registers of the microprocessor. This chapter is organized as follows:

- Ibox internal processor registers
- Abox internal processor registers
- PAL_TEMP internal processor registers
- Register reset state

4.2 Ibox Internal Processor Registers

This section describes each Ibox internal processor register (IPR).

4.2.1 Translation Buffer Tag Register (TB_TAG)

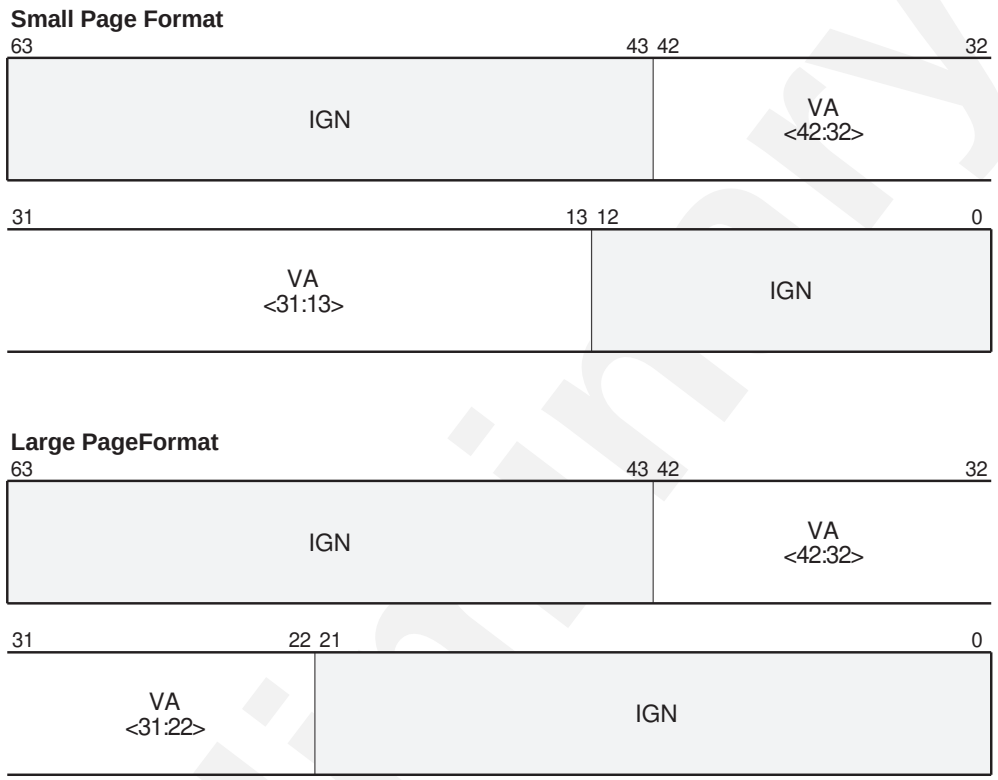
This write-only register holds the tag for the next translation buffer update operation in the instruction translation buffer (ITB) or the data translation buffer (DTB). The tag is written to a temporary register and not transferred to the ITB or DTB until the Instruction Translation Buffer Page Table Entry (ITB_PTE) or the Data Translation Buffer Page Table Entry (DTB_PTE) register is written. The entry to be written is chosen at the time of the ITB_PTE or DTB_PTE write operation. The replacement algorithm (a not-last-used algorithm or a round-robin algorithm) can be chosen for the DTB using ABOX_CTL bit 7. Figure 4–1 shows the TB_TAG register format.

Note

Writing to the Instruction Translation Buffer Tag register (ITB_TAG) is only performed while in PALmode, regardless of the state of the hardware enable (HWE) bit in the ICCSR register.

Preliminary

Figure 4–1 Translation Buffer Tag Register



4.2.2 Instruction Translation Buffer Page Table Entry Register (ITB_PTE)

This read/write register represents 12 page table entries split into two distinct arrays. The first eight page table entries provide small page (8K byte) translations and the remaining four provide large page (4 MB) translations. The entry to be written is chosen by a not-last-used algorithm implemented in hardware for each array independently and the status of the TB_CTL. Writes to the ITB_PTE register use the memory format bit positions as described in the Alpha Architecture Handbook, with the exception that some fields are ignored.

The ITB's tag array is updated simultaneously from the Internal Tag register when the ITB_PTE register is written. ITB_PTE register reads require two instructions. The first instruction sends the PTE data to the Instruction Translation Buffer Page Table Entry Temporary register (ITB_PTE_TEMP)

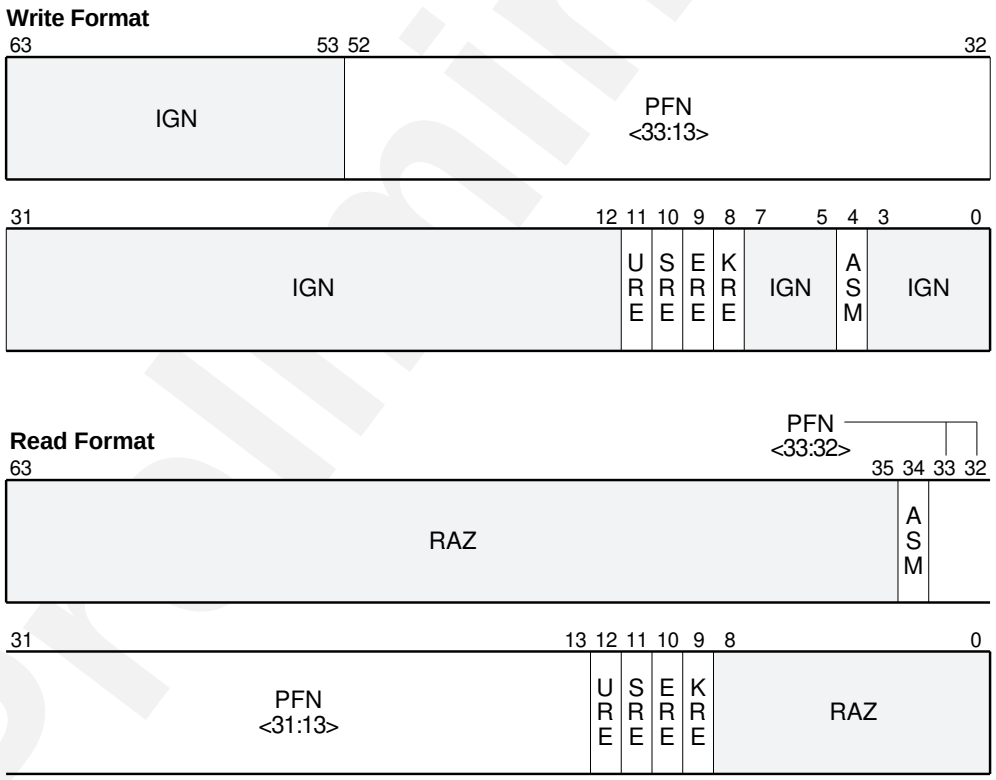
Preliminary

and the second instruction, reading from the ITB_PTE_TEMP register, returns the PTE entry to the register file. Reading or writing the ITB_PTE register increments the TB entry pointer corresponding to the large/small page selection indicated by the TB_CTL, which allows reading the entire set of ITB_PTE register entries. Figure 4–2 shows the ITB_PTE register format.

Note

Reading and writing the ITB_PTE register is only performed while in PALmode regardless of the state of the HWE bit in the ICCSR IPR.

Figure 4–2 Translation Buffer Page Table Entry Register



Preliminary

4.2.3 Instruction Cache Control and Status Register (ICCSR)

The ICCSR register contains various Ibox hardware enables. The only architecturally defined bit in this register is the floating point enable (FPE), which enables floating point instructions. When cleared, all floating point instructions generate FEN exceptions. Most of this register is cleared by hardware at reset. Fields that are not cleared at reset include ASN, PC0, and PC1.

The hardware enable bit allows the special privileged architecture library (PAL) instructions to execute in kernel mode. This bit is intended for diagnostic or operating system alternative PAL routines only. It does not allow access to the ITB registers if not running in PALmode. Figure 4–3 shows the ICCSR register format.

Table 4–1 lists the ICCSR register fields and a brief description. Table 4–2 lists BPE and BHE branch selection.

Preliminary

Figure 4–3 Instruction Cache Control and Status Register (ICCSR)

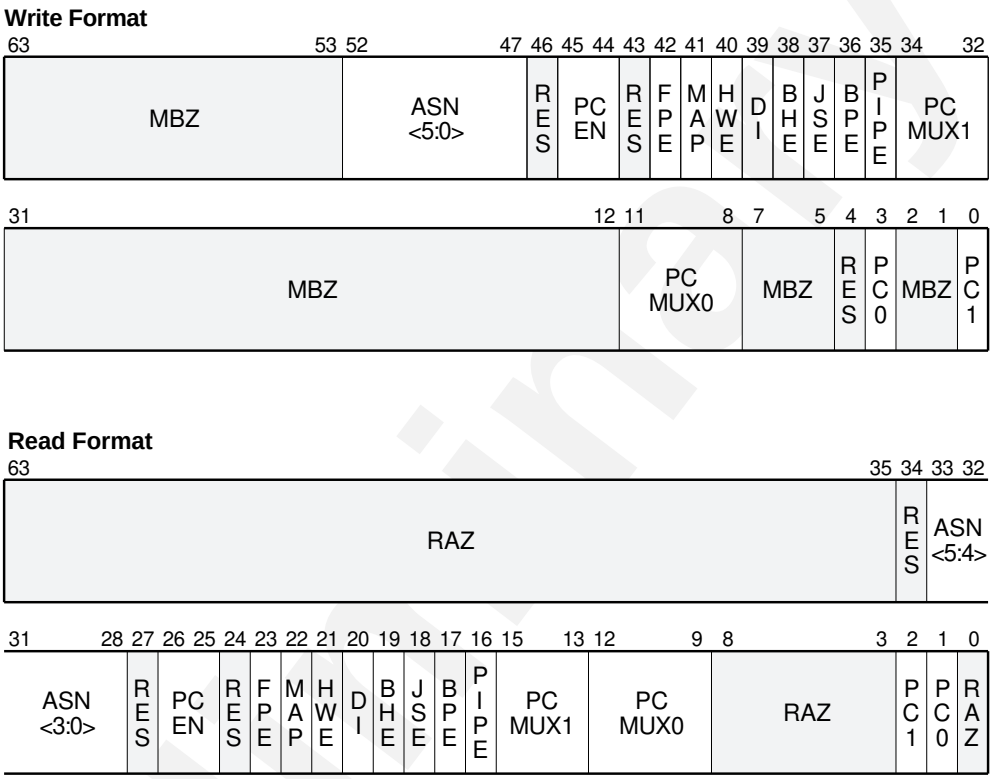


Table 4–1 Instruction Cache Control and Status Register (ICCSR) Bit Description

Field	Type	Description
FPE	RW,0	If set, floating point instructions can be issued. If clear, floating point instructions cause FEN exceptions.
HWE	RW,0	If set, allows the five PALRES instructions to be issued in kernel mode. If clear, attempts to execute PALRES instructions while not in PALmode result in OPCDEC exceptions.

(continued on next page)

Preliminary

Table 4–1 (Cont.) Instruction Cache Control and Status Register (ICCSR) Bit Description

Field	Type	Description
MAP	RW,0	If set, allows superpage I-stream memory mapping of VP <33:13> directly to physical PC <33:13> essentially bypassing ITB for VPC addresses containing VPC <42:41> = 2. Superpage mapping is allowed in kernel mode only. The Icache ASM bit is always set. If clear, superpage mapping is disabled.
DI	RW,0	If set, enables dual issue. If clear, instructions can only single issue.
BHE	RW,0	Used in conjunction with BPE. See Table 4–2 for programming information.
JSE	RW,0	If set, enables the JSR stack to push a return addresses. If clear, JSR stack is disabled.
BPE	RW,0	Used in conjunction with BHE. See Table 4–2 for programming information.
PIPE	RW,0	If clear, causes all hardware interlocked instructions to drain the machine and waits for the write buffer to empty before issuing the next instruction. Examples of instructions that do not cause the pipe to drain include HW_MTPR, HW_REI, conditional branches, and instructions that have a destination register of R31. If set, pipeline proceeds normally.
PCMUX1	RW,0	See Table 4–3 for programming information.
PCMUX0	RW,0	See Table 4–4 for programming information.
PC1	RW	If clear, enables performance counter 1 interrupt request after 2^{12} events counted. If set, enables performance counter 1 interrupt request after 2^{8} events counted.
PC0	RW	If clear, enables performance counter 0 interrupt request after 2^{16} events counted. If set, it enables performance counter 0 interrupt request after 2^{12} events counted.
PCEN	RW,0	If either bit is set, both performance counters increment, if both bits are clear, performance counters are disabled.
ASN	RW	The ASN field is used in conjunction with the Icache to further qualify cache entries and avoid some cache flushes. The ASN is written to the Icache during fill operations and compared with the I-stream data on fetch operations. Mismatches invalidate the fetch without affecting the Icache.
RES	RW,0	Reserved by Digital and should not be used by software.

Preliminary

Caution

Using the HW_MTPR instruction to update the EXC_ADDR register while in the native mode is restricted to bit 0 being equal to 0. The combination of the native mode and EXC_ADDR bit 0 being equal to one causes UNDEFINED behavior.

Table 4–2 BHE, BPE Branch Prediction Selection

BPE	BHE	Prediction
0	X	Not taken
1	0	Sign of displacement
1	1	Branch history table

4.2.3.1 Performance Counters

The performance counters are reset to zero upon powerup. Otherwise, they are never cleared. The counters are intended as a means of counting events over a long period of time, relative to the event frequency. They provide no means of extracting intermediate counter values.

Since the counters continuously accumulate selected events, despite interrupts being enabled, the first interrupt after selecting a new counter input has an error bound as large as the selected overflow range. Some inputs can over count events occurring simultaneously with D-stream errors that abort the actual event very late in the pipeline. For example, when counting load instructions, attempts to execute a load resulting in a TB miss exception will increment the performance counter after the first aborted execution attempt and again after the TB fill routine when the load instruction reissues and completes.

Performance counter interrupts are reported six cycles after the event that caused the counter to overflow. Additional delay can occur before an interrupt is serviced, if the processor is executing PALcode that always disables interrupts. Events occurring during the interval between counter overflow and interrupt service are counted toward the next interrupt. Only in the case of a complete counter wrap-around while interrupts are disabled will an interrupt be missed.

The six cycles before an interrupt is triggered implies that a maximum of 12 instructions may have completed before the start of the interrupt service routine.

Preliminary

When counting Icache misses, no intervening instructions can complete and the exception PC contains the address of the last Icache miss. Branch mispredictions allow a maximum of only two instructions to complete before start of the interrupt service routine.

Table 4–3 lists performance counter 0 inputs and Table 4–4 lists performance counter 1 inputs.

Table 4–3 Counter 0 Input Selection

MUX0 <3:0>	Input	Comments
000X	Total Issues ÷2	Counts total issues divided by 2, dual issue increments count by 1.
000X	Pipeline Dry	Counts cycles where nothing issued due to lack of valid I-stream data. Causes include Icache fill, mispredictions, branch delay slots, and pipeline drain for exception.
010X	Load Instructions	Count all Load instructions.
011X	Pipeline Frozen	Counts cycles where nothing issued due to resource conflict.
100X	Branch Instructions	Counts all conditional branches, unconditional branches, JSR, and HW_REI instructions.
1011	PALmode	Counts cycles while executing in PALmode.
1010	Total Cycles	Counts total cycles.
110X	Total Nonissues÷2	Counts total nonissues divided by 2; that is, no issue increments count by 1.
111X	Memory controller register EAR	Counts external interface events specified in the memory controller EAR register. Refer to memory controller chapter for the events that can be monitored.

Table 4–4 Counter 1 Input Selection

MUX <2:0>	Input	Comments
000	Dcache miss	Counts total Dcache misses.
001	Icache miss	Counts total Icache misses.

(continued on next page)

Preliminary

Table 4–4 (Cont.) Counter 1 Input Selection

MUX <2:0>	Input	Comments
010	Dual issues	Counts cycles of dual issue.
011	Branch mispredicts	Counts both conditional branch mispredictions and JSR or HW_REI mispredictions. Conditional branch mispredictions cost 4 cycles and others cost 5 cycles of dry pipeline delay.
100	FP instructions	Counts total floating point operate instructions, that is, no FP branch, load, or store.
101	Integer operate	Counts integer operate instructions including LDA and LDAH with destination other than R31.
110	Store instructions	Counts total store instructions.
111	Memory controller register EAR	Counts external interface events specified in the memory controller EAR register. Refer to memory controller chapter for the events that can be monitored.

4.2.4 Instruction Translation Buffer Page Table Entry Temporary Register (ITB_PTE_TEMP)

The ITB_PTE_TEMP register is a read-only holding register for ITB_PTE read data. Reads of ITB_PTE register require two instructions to return data to the register file. The two instructions are as follows:

1. Read the ITB_PTE register data to the ITB_PTE_TEMP register.
2. Read the ITB_PTE_TEMP register data to the integer register file.

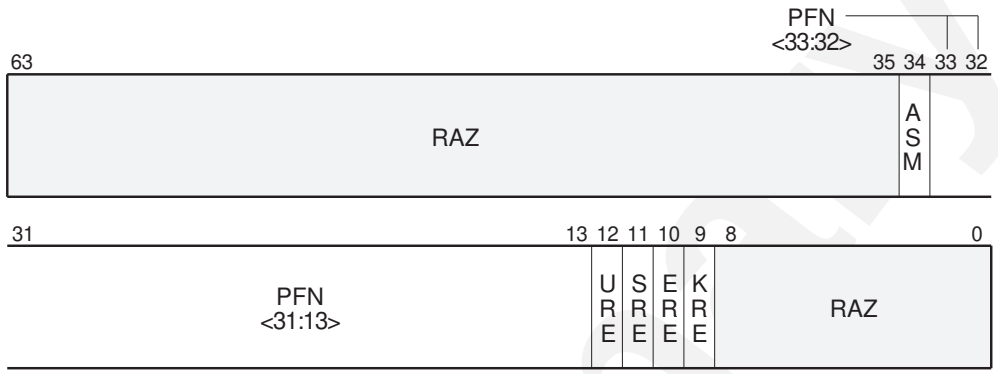
The ITB_PTE_TEMP register is updated on all ITB accesses, both read and write. A read of the ITB_PTE to the ITB_PTE_TEMP should be followed closely by a read of the ITB_PTE_TEMP to the register file. Figure 4–4 shows the ITB_PTE_TEMP register format.

Note

Reading the ITB_PTE_TEMP register is only performed while in PALmode regardless of the state of the HWE bit in the ICCSR.

Preliminary

Figure 4–4 ITB_PTE_TEMP Register



4.2.5 Exceptions Address Register (EXC_ADDR)

The EXC_ADDR register is a read/write register used to restart the system after exceptions or interrupts. The register can be read and written by the software, by way of the HW_MTPR instruction. Also, the EXC_ADDR can be written directly to by the hardware.

The HW_REI instruction executes a jump to the address contained in the EXC_ADDR register. The EXC_ADDR register is written by hardware after an exception to provide a return address for PALcode. The instruction pointed to by the EXC_ADDR register did not complete its execution. The LSB of the EXC_ADDR register is used to indicate PALmode to the hardware. When the LSB is clear, the HW_REI instruction executes a jump to native (nonPAL) mode, enabling address translation.

CALL_PAL exceptions load the EXC_ADDR with the PC of the instruction following the CALL_PAL. This function allows CALL_PAL service routines to return without needing to increment the value in the EXC_ADDR register. This feature requires careful treatment in PALcode. Arithmetic traps and machine check exceptions can prompt CALL_PAL exceptions resulting in an incorrect value being saved in the EXC_ADDR register. In the cases of an arithmetic trap or machine check exception (only in these cases), EXC_ADDR bit 1 takes on special meaning. PALcode servicing these two exceptions must:

- Interpret a 0 in EXC_ADDR bit 1 as indicating that the PC in EXC_ADDR <63:2> is too large by a value of 4 bytes and subtract 4 before executing a HW_REI from this address.
- Interpret a 1 in EXC_ADDR bit 1 as indicating that the PC in EXC_ADDR <63:2> is correct and clear the value of EXC_ADDR bit 1

Preliminary

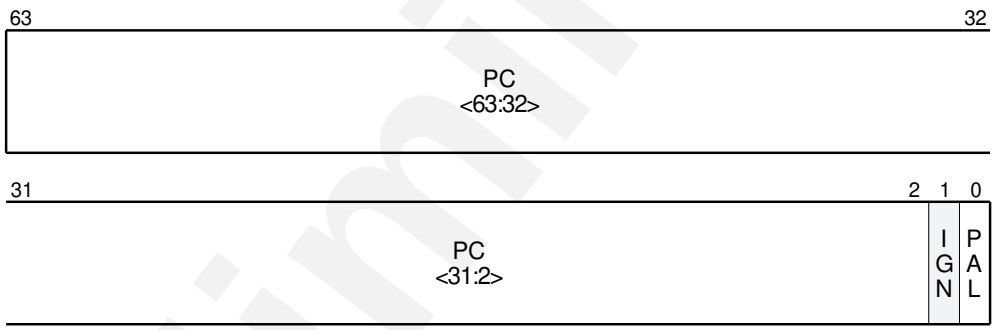
All other PALcode entry points except reset can expect EXC_ADDR bit 1 to be 0.

The logic allows the following code sequence to conditionally subtract 4 from the address in the EXC_ADDR register without the use of an additional register. This code sequence must be present in arithmetic trap and machine check flows only.

```
HW_MFPR Rx, EXC_ADDR    ; read EXC_ADDR into GPR
SUBQ    Rx, 2,Rx        ; subtract 2 causing borrow if bit [1]=0
BIC     Rx, 2,Rx        ; clear bit [1]
HW_MTPR Rx, EXC_ADDR    ; write back to EXC_ADDR
```

Figure 4–5 shows the exception address register format.

Figure 4–5 Exception Address Register



4.2.6 Clear Serial Line Interrupt Register (SL_CLR)

The SL_CLR is a write-only register that clears the:

- Serial line interrupt request
- Performance counter interrupt requests

The indicated bit must be written with a zero to clear the selected interrupt source. Figure 4–6 shows the clear serial line interrupt register format. Table 4–5 lists the register fields and a description.

Preliminary

Figure 4–6 Clear Serial Line Interrupt Register (SL_CLR)

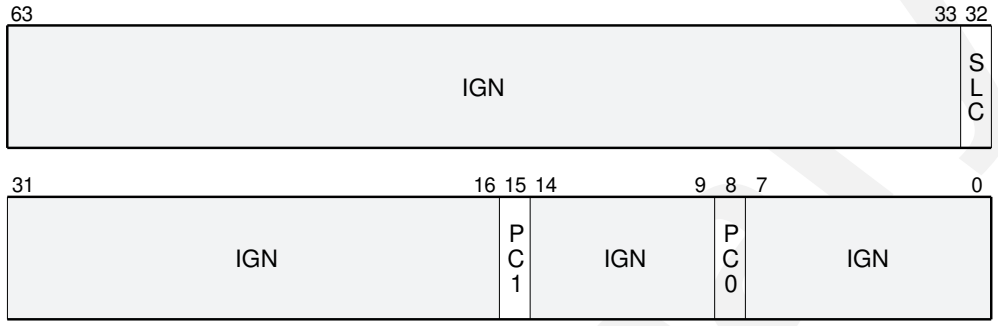


Table 4–5 Clear Serial Line Interrupt Register (SL_CLR) Bit Description

Bits	Field	Type	Description
15	PC1	W0C	Clears the performance counter 1 interrupt request
08	PC0	W0C	Clears the performance counter 0 interrupt request
32	SLC	W0C	Clears the serial line interrupt request

4.2.7 Serial Line Receive Register (SL_RCV)

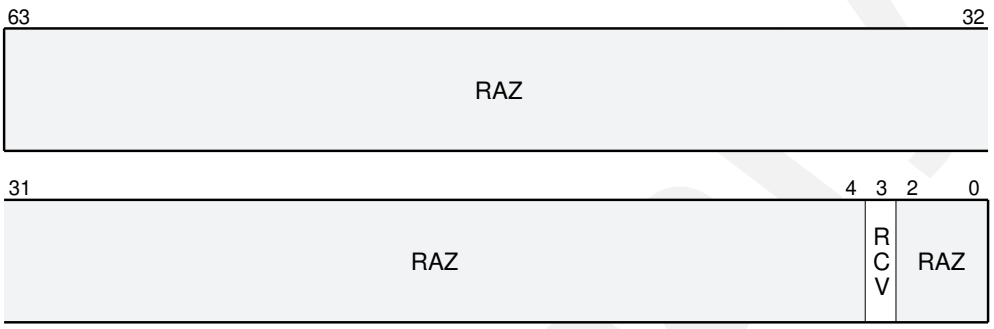
The SL_RCV register contains a single read-only bit (RCV). This bit is used with the interrupt control registers, the sRomD_h pin, and the sRomClk_h pin to provide an on-chip serial line function. The RCV bit is functionally connected to the sRomD_h pin after the Icache is loaded from the external serial ROM. Using a software timing loop, the RCV bit can be read to receive external data one bit at a time.

A serial line interrupt is requested on detection of any transition on the receive line that sets the SL_REQ bit in the HIRR. The serial line interrupt can be disabled by clearing the HIER register SL_ENA bit.

Figure 4–7 shows the Serial Line Receive Register format.

Preliminary

Figure 4–7 Serial Line Receive Register



4.2.8 Instruction Translation Buffer ZAP Register (ITBZAP)

A write to this register invalidates all twelve instruction translation buffer (ITB) entries. It also resets both the NLU pointers to their initial state. The ITBZAP register is only written to in PALmode.

4.2.9 Instruction Translation Buffer ASM Register (ITBASM)

A write to this register invalidates all ITB entries, in which the ASM bit i equal to zero. The ITBASM register is only written to in PALmode.

4.2.10 Instruction Translation Buffer IS Register (ITBIS)

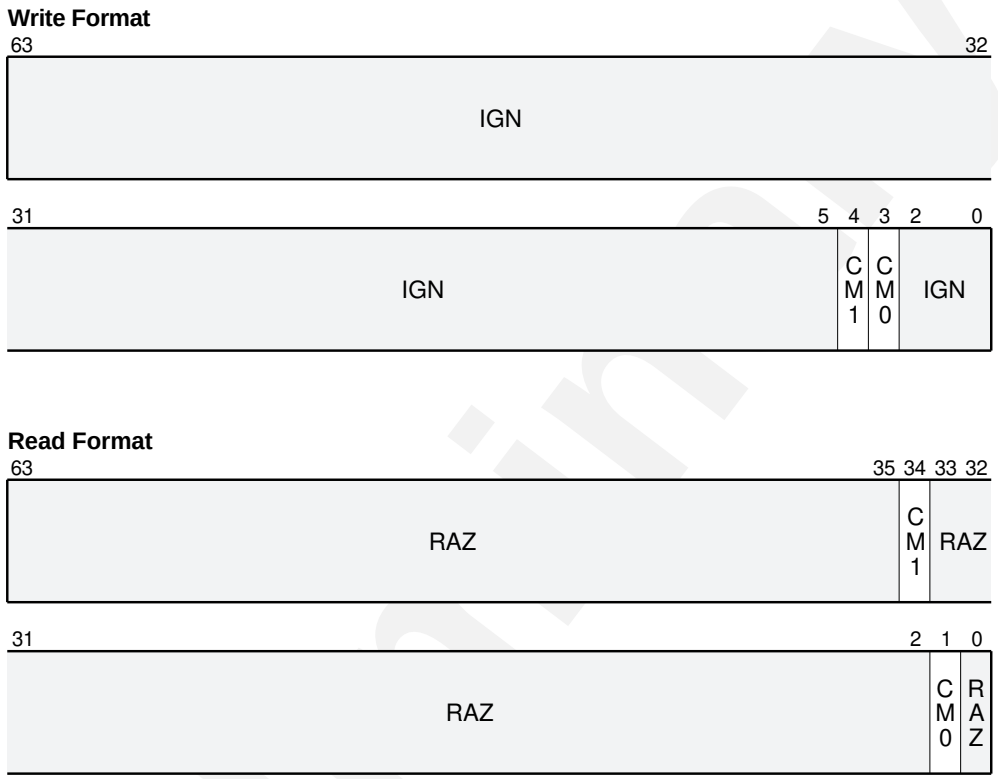
A write to the ITBIS register invalidates all twelve ITB entries. It also resets both the NLU pointers to their initial state. The ITBIS register is only written to in PALmode.

4.2.11 Processor Status Register (PS)

The PS register is a read/write register containing only the current mode bits of the architecturally defined PS. Figure 4–8 shows the register format.

Preliminary

Figure 4–8 Processor Status Register



4.2.12 Exception Summary Register (EXC_SUM)

The EXC_SUM register records the various types of arithmetic traps that occurred since the last time the EXC_SUM was written (cleared). When the result of an arithmetic operation produces an arithmetic trap, the corresponding EXC_SUM bit is set.

The register containing the result of the operation is recorded in the Exception Register Write Mask register, as a single bit in a 64-bit field specifying registers F31-F0 and I31-I0. The EXC_SUM register provides a one-bit window to the Exception Register Write Mask register. This is visible only through the EXC_ SUM register.

Each read to the EXC_SUM shifts one bit in order F31-F0 then I31-I0. The read also clears the corresponding bit. The EXC_SUM must be read 64 times to extract the complete mask and clear the entire register.

Preliminary

Any write to EXC_SUM clears bits <8:2> and does not affect the write mask bit.

The Write Mask register bit clears three cycles after a read. Code intended to read the register must allow at least three cycles between reads. This allows the clear and shift operations to complete in order to ensure reading successive bits. Figure 4–9 shows the exception summary register format. Table 4–6 lists the register Bit Descriptions.

Figure 4–9 Exception Summary Register (EXC_SUM)

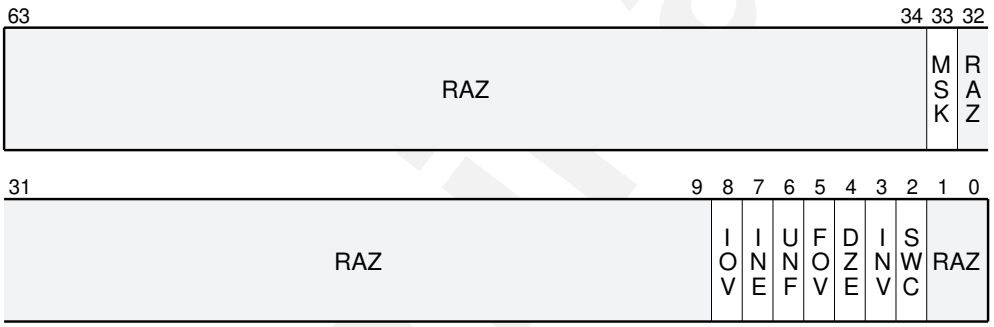


Table 4–6 Exception Summary Register (EXC_SUM) Bit Description

Bits	Field	Type	Description
02	SWC	WA	Indicates software completion possible. The bit is set after a floating point instruction containing the /S modifier completes with an arithmetic trap and all previous floating point instructions that trapped since the last HW_ MTPR EXC_SUM also contained the /S modifier. The SWC bit is cleared whenever a floating point instruction without the /S modifier completes with an arithmetic trap. The bit remains cleared regardless of additional arithmetic traps until the register is written by way of an HW_MTPR instruction. The bit is always cleared upon any HW_MTPR write to the EXC_SUM register.
03	INV	WA	Indicates invalid operation.
04	DZE	WA	Indicates divide by zero.
05	FOV	WA	Indicates floating point overflow.

(continued on next page)

Preliminary

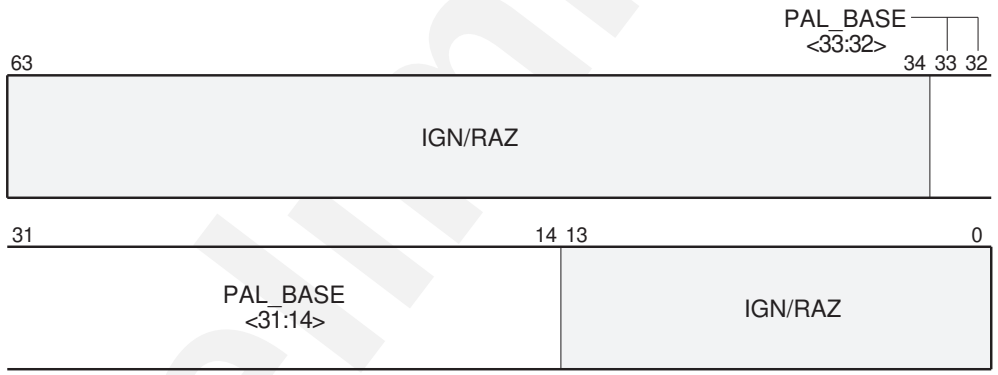
Table 4–6 (Cont.) Exception Summary Register (EXC_SUM) Bit Description

Bits	Field	Type	Description
06	UNF	WA	Indicates floating point underflow.
07	INE	WA	Indicates floating inexact error.
08	IOV	WA	Indicates Fbox convert to integer overflow or integer arithmetic overflow.
34	MSK	RC	Exception Register Write Mask IPR window.

4.2.13 PAL Base Address Register (PAL_BASE)

The PAL_BASE register is a read/write register containing the base address for PALcode. This register is cleared by the hardware at reset. Figure 4–10 shows the PAL base address register format.

Figure 4–10 PAL Base Address Register



4.2.14 Hardware Interrupt Request Register (HIRR)

The HIRR is a read-only register providing a record of all currently outstanding interrupt requests and summary bits at the time of the read. For each bit of the HIRR, there is a corresponding bit of the Hardware Interrupt Enable register (HIER) that must be set to request an interrupt. In addition to returning the status of the hardware interrupt requests, a read of the HIRR returns the state of the software interrupt and AST requests.

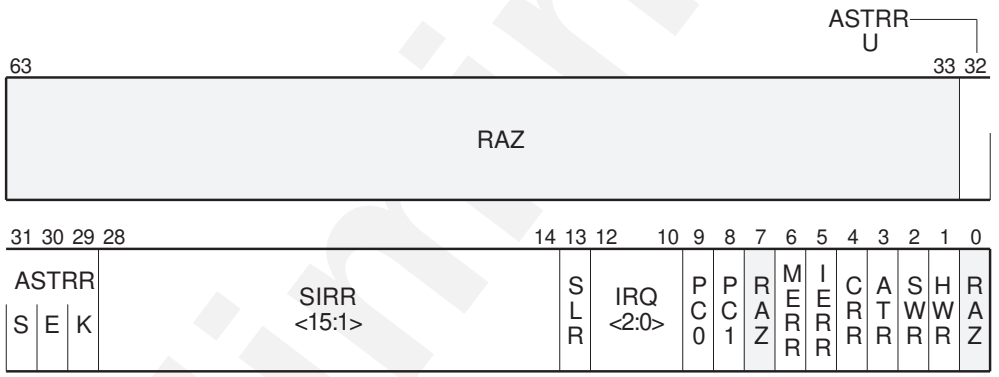
Preliminary

Note

A read of the HIRR can return a value of zero if the hardware interrupt was released before the read (passive release).

The register guarantees that the HWR bit reflects the status as shown by the HIRR bits. All interrupt requests are blocked while executing in PALmode. Figure 4–11 shows the hardware interrupt request register format. Table 4–7 lists the register fields and gives a description of each. For additional information on interrupt operations, refer to Section 2.2.4.

Figure 4–11 Hardware Interrupt Request Register (HIRR)



Note

The bit descriptions in Table 4–7 apply to the request bits in the HIRR, and also to the corresponding request or enable bits in the HIER, SIRR, SIER, ASTRR, and ASTER.

Table 4–7 Hardware Interrupt Request Register (HIRR) Bit Description

Bits	Field	Type	Description
01	HWR	RO	Set if any hardware interrupt request and corresponding enable is set.

(continued on next page)

Preliminary

Table 4–7 (Cont.) Hardware Interrupt Request Register (HIRR) Bit Description

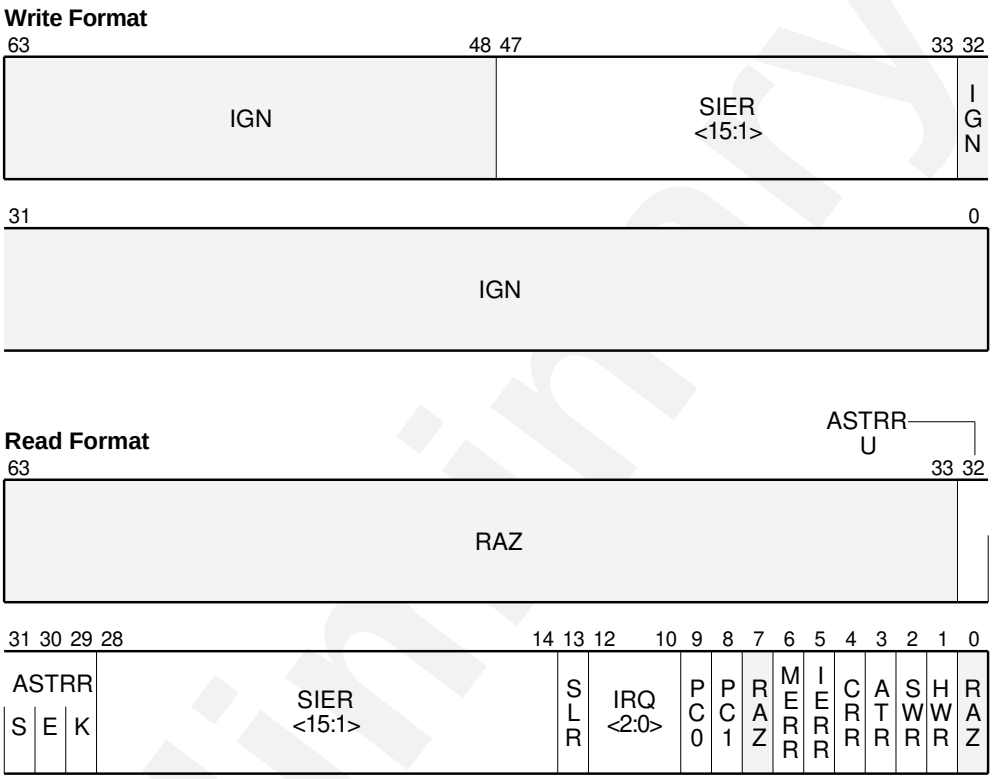
Bits	Field	Type	Description
02	SWR	RO	Set if any software interrupt request and corresponding enable is set.
03	ATR	RO	Set if any AST request and corresponding enable is set. This bit also requires that the processor mode be equal to or higher than the request mode. SIER 2 must be set to allow AST interrupt requests.
04	CRR	RO	CRD correctable read error interrupt request. This interrupt is cleared by way of the SL_CLR register.
5	IERR	RO	IOC error interrupt request. This interrupt is cleared by way of the IOC_STAT0 (Section 6.5.6).
6	MERR	RO	Memory controller error interrupt requests. This interrupt is cleared by way of the memory controller ESR register.
08	PC1	RO	Performance counter 1 interrupt request.
09	PC0	RO	Performance counter 0 interrupt request.
12:10	IRQ<2:0>	RO	Corresponds to pins IRQ_H<2:0>.
13	SLR	RO	Serial line interrupt request. Also see SL_RCV, SL XMIT, and SL_CLR.
28:14	SIRR <15:1>	RO	Corresponds to software interrupt request 15 through 1.
32:29	ASTRR	RO	Corresponds to AST request 3 through 0 (USEK).

4.2.15 Software Interrupt Request Register (SIRR)

The SIRR is a read/write register used to control software interrupt requests. For each bit of the SIRR, there is a corresponding bit of the Software Interrupt Enable register (SIER) that must be set to request an interrupt. Reads of the SIRR return the complete set of interrupt request registers and summary bits (see Table 4–7 for details). All interrupt requests are blocked while executing in PALmode. Figure 4–12 shows the SIRR format.

Preliminary

Figure 4–12 Software Interrupt Request Register



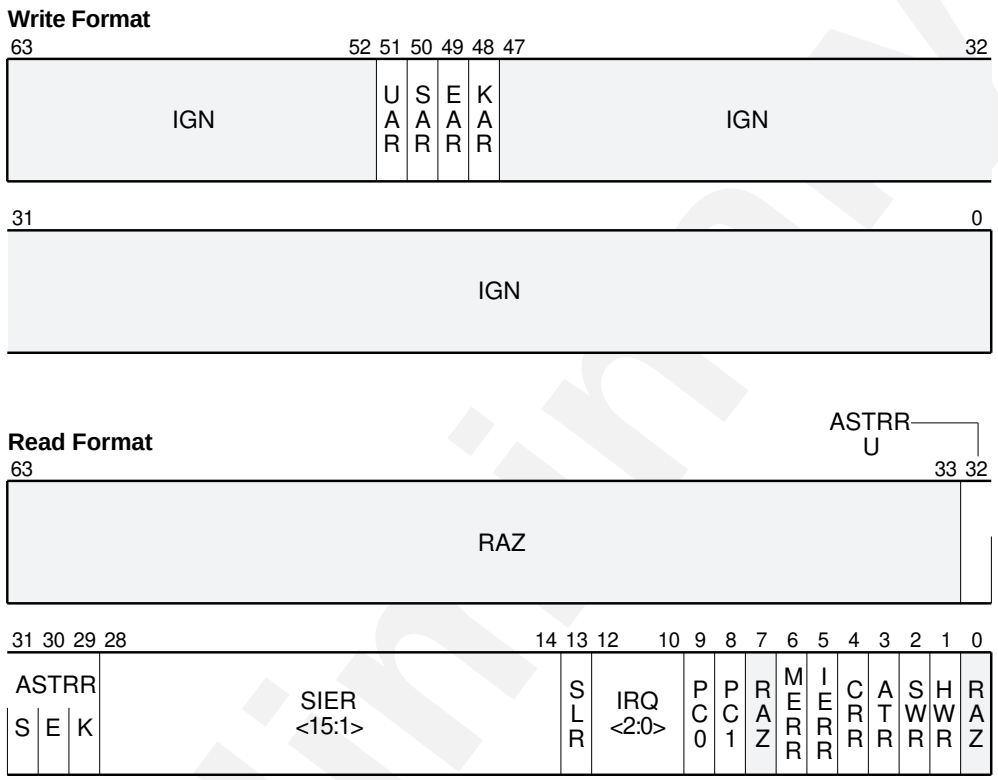
4.2.16 Asynchronous Trap Request Register (ASTRR)

The ASTRR is a read/write register. It contains bits to request AST interrupts in each of the processor modes. In order to generate an AST interrupt, the corresponding enable bit in the ASTER must be set. Also, the processor must be in the selected processor mode or higher privilege as described by the current value of the PS CM bits. AST interrupts are enabled if the SIER 2 is set. This provides a mechanism to lock out AST requests over certain IPL levels.

All interrupt requests are blocked while executing in PALmode. Reads of the ASTRR return the complete set of interrupt request registers and summary bits. See Table 4–7 for details. Figure 4–13 shows the ASTRR format.

Preliminary

Figure 4–13 Asynchronous Trap Request Register



4.2.17 Hardware Interrupt Enable Register (HIER)

The HIER is a read/write register. It is used to enable corresponding bits of the HIRR requesting interrupt. The PC0, PC1, SLE, and CRE bits of this register enable the:

- Performance counters
- Serial line
- Correctable read interrupts

There is a one-to-one correspondence between the interrupt requests and enable bits. As with the reads of the interrupt request registers, reads of the HIER return the complete set of interrupt enable registers. See Table 4–7 for details. Figure 4–14 shows the hardware interrupt enable register format. Table 4–8 lists the register fields and a description of each.

Preliminary

Figure 4–14 Hardware Interrupt Enable Register (HIER)

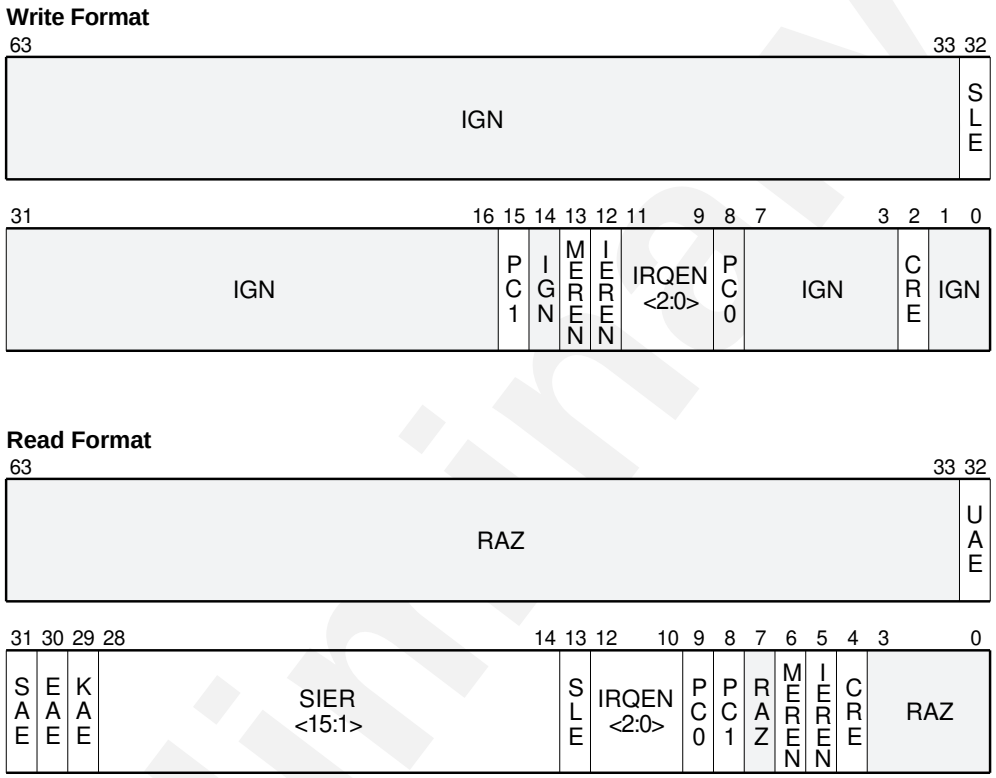


Table 4–8 Hardware Interrupt Enable Register (HIER) Bit Description

Bits	Field	Type	Description
32:29	ASTER <3:0>	RW	Corresponds to ASTRR enable 3 through 0 (USEK).
28:14	SIER <15:1>	RW	Corresponds to software interrupt requests 15 through 1.
13	SLE	RW	Serial line interrupt enable. Also see SL_RCV, SL_ XMIT, and SL_CLR.
12:10	IRQEN<2:0>	RW	Interrupt enables for pins IRQ_H<2:0>.
09	PC0	RW	Performance counter 0 interrupt enable.

(continued on next page)

Preliminary

Table 4–8 (Cont.) Hardware Interrupt Enable Register (HIER) Bit Description

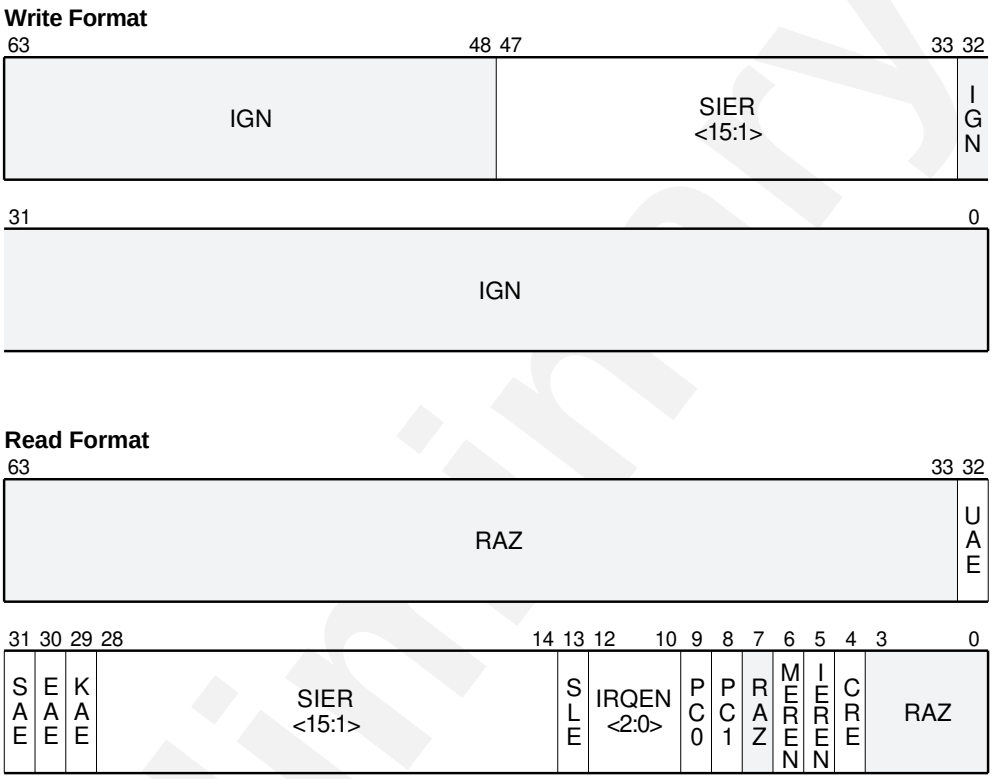
Bits	Field	Type	Description
08	PC1	RW	Performance counter 1 interrupt enable.
6	MEREN	RW	Memory controller error interrupt enable. Also see Memory controller EAR.
5	IEREN	RW	IOC error interrupt enables Also see IOC_STAT0 register.
04	CRE	RW	CRD Correctable read error interrupt enable. This interrupt request is cleared by way of the SL_CLR register.

4.2.18 Software Interrupt Enable Register (SIER)

The SIER is a read/write register. It is used to enable corresponding bits of the SIRR requesting interrupts. There is a one-to-one correspondence between the interrupt requests and enable bits. As with the reads of the interrupt request registers, reads of the SIER return the complete set of interrupt enable registers. See Table 4–7 for details. Figure 4–15 shows the software interrupt enable register format.

Preliminary

Figure 4–15 Software Interrupt Enable Register

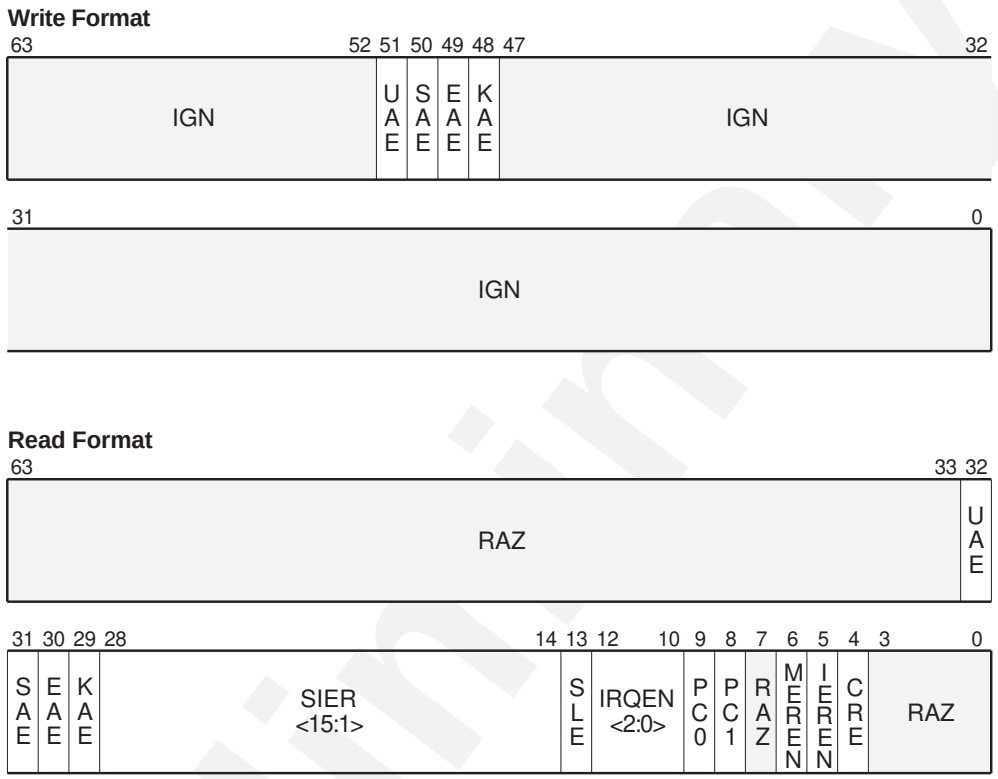


4.2.19 AST Interrupt Enable Register (ASTER)

The ASTER is a read/write register. It is used to enable corresponding bits of the ASTRR requesting interrupts. There is a one-to-one correspondence between the interrupt requests and enable bits. As with the reads of the interrupt request registers, reads of the ASTER return the complete set of interrupt enable registers. See Table 4–7 for details. Figure 4–16 shows the ASTER format.

Preliminary

Figure 4–16 AST Interrupt Enable Register

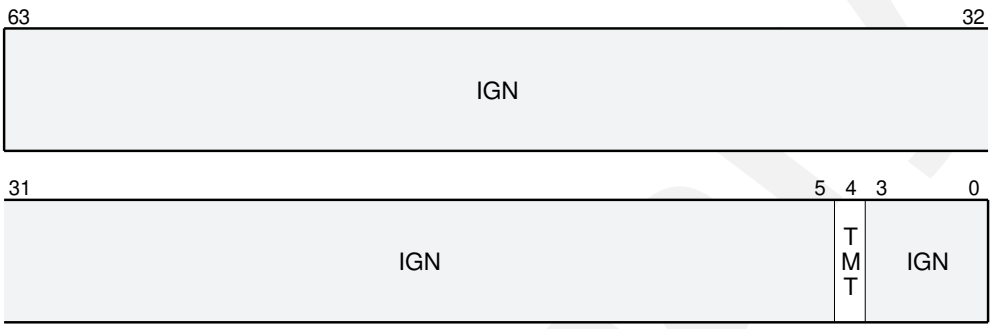


4.2.20 Serial Line Transmit Register (SL_XMIT)

The SL_XMIT register contains a single write-only bit. This bit is used with the interrupt control registers, the sRomD_h pin, and the sRomClk_h pin to provide an on-chip serial line function. The TMT bit is functionally connected to the sRomClk_h pin after the Icache is loaded from the external serial ROM. Writing the TMT bit can be used to transmit data off chip, one bit at a time under a software timing loop. Figure 4–17 shows the SL_XMIT register format.

Preliminary

Figure 4–17 Serial Line Transmit Register



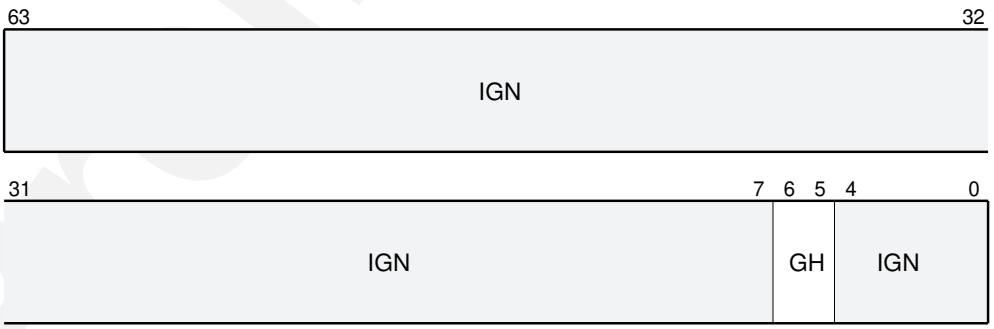
4.3 Abox Internal Processor Registers

The following sections describe the Abox internal processor registers.

4.3.1 Translation Buffer Control Register (TB_CTL)

The granularity hint (GH) field selects between the TB page mapping sizes. There are two sizes in the ITB and four sizes in the DTB. When only two sizes are provided, the large-page-select (GH=11(bin)) field selects the largest mapping size (512 * 8 KB). All other values select the smallest (8 KB) size. The GH field affects both reads and writes to the ITB and DTB. Figure 4–18 shows the translation buffer control register format.

Figure 4–18 Translation Buffer Control Register



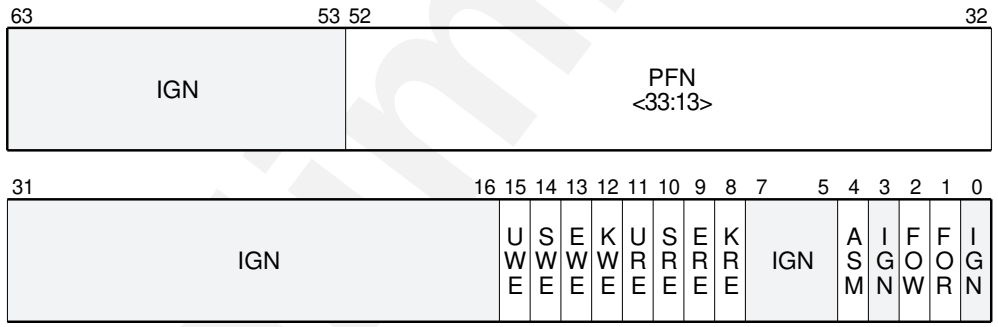
Preliminary

4.3.2 Data Translation Buffer Page Table Entry Register (DTB_PTE)

The DTB_PTE register is a read/write register representing the 32-entry DTB. The entry to be written is chosen by a not-last-used (NLU) algorithm implemented in the hardware. Writes to the DTB_PTE use the memory format bit positions as described in the Alpha Architecture Handbook with the exception that some fields are ignored. The valid bit is not represented in hardware.

The DTB's tag array is updated simultaneously from the internal tag register when the DTB_PTE register is written. Reads of the DTB_PTE require two instructions. The first instruction sends the PTE data to the data translation buffer page table entry temporary register (DTB_PTE_TEMP). The second instruction, reading from the DTB_PTE_TEMP register, returns the PTE entry to the register file. Reading or writing the DTB_PTE register increments the TB entry pointer of the DTB, which allows reading the entire set of DTB_PTE entries. Figure 4–19 shows the DTB_PTE register format.

Figure 4–19 Data Translation Buffer Page Table Entry Register



4.3.3 Data Translation Buffer Page Table Entry Temporary Register (DTB_PTE_temp)

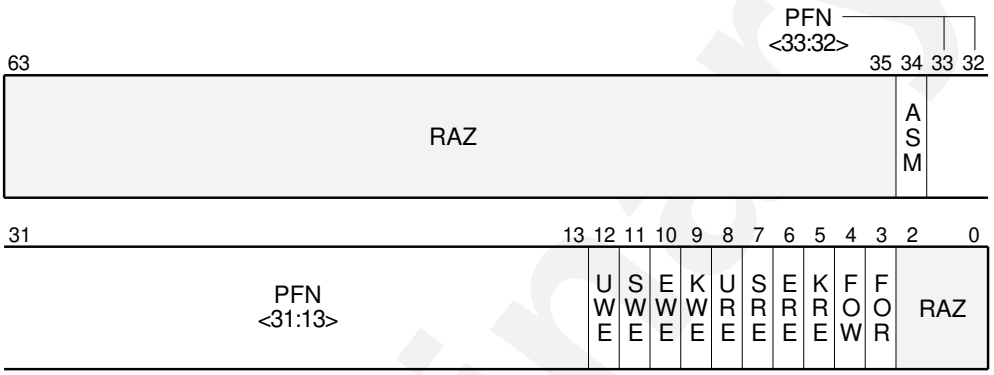
The DTB_PTE_TEMP register is a read-only holding register for DTB_PTE read data. Reads of the DTB_PTE require two instructions to return the data to the register file:

- Read the DTB_PTE register data to the DTB_PTE_TEMP register.
- Read the DTB_PTE_TEMP register data to the integer register file.

Preliminary

Figure 4–20 shows DTB_PTE_TEMP register format.

Figure 4–20 Data translation Buffer Page table Entry Temporary Register



4.3.4 Memory Management Control and Status Register (MM_CSR)

When D-stream faults occur the information about the fault is latched and saved in the MM_CSR register. The virtual address register (VA) and MM_ CSR registers are locked against further updates until the software reads the Virtual Address register. PALcode must explicitly unlock this register whenever its entry point is higher in priority than a DTB miss. The MM_CSR bits are only modified by the hardware when the register is not locked and a memory management error or a DTB miss occurs. The MM_CSR is unlocked after reset. Figure 4–21 shows the MM_CSR register format. Table 4–9 lists the register fields and a brief description.

Preliminary

Figure 4–21 Memory Management Control and Status Register (MM_CSR)

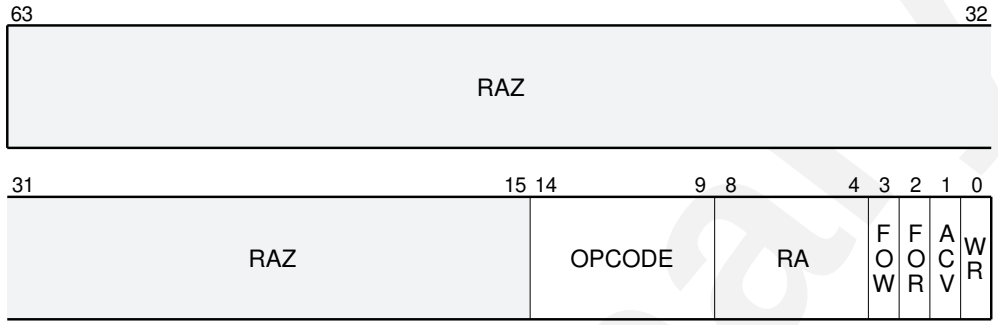


Table 4–9 Memory Management Control and Status Register (MM_CSR) Bit Description

Bits	Field	Type	Description
0	WR	RO	Set if reference that caused error was a write.
1	ACV	RO	Set if reference caused an access violation.
02	FOR	RO	Set if reference was a read and the PTE's FOR bit was set.
03	FOW	RO	Set if reference was a write and the PTE's FOW bit was set.
8:4	RA	RO	Ra field of the faulting instruction.
14:9	OPCODE	RO	Opcode field of the faulting instruction.

4.3.5 Virtual Address Register (VA)

When D-stream faults or DTB misses occur, the effective virtual address associated with the fault or miss is latched in the read-only VA register. The VA and MM_CSR registers are locked against further updates until the software reads the VA register. The VA register is unlocked after reset. PALcode must explicitly unlock VA whenever its entry point is higher in priority than a DTB miss.

4.3.6 Data Translation Buffer ZAP Register (DTBZAP)

The DTBZAP is a pseudo-register. Any write to this register invalidates all 32 DTB entries. It also resets the not-last-used (NLU) pointer to its initial state.

Preliminary

4.3.7 Data Translation Buffer ASM Register (DTBASM)

The DTBASM is a pseudo-register. Any write to this register invalidates all 32 DTB entries in which the ASM bit is equal to zero.

4.3.8 Data Translation Buffer Invalidate Single Register (DTBIS)

Any write to this pseudo-register will invalidate the DTB entry, which maps the virtual address held in the integer register. The integer register is identified by the Rb field of the HW_MTPR instruction, used to perform the write.

4.3.9 Flush Instruction Cache Register (FLUSH_IC)

Any write to this pseudo-register flushes the entire instruction cache.

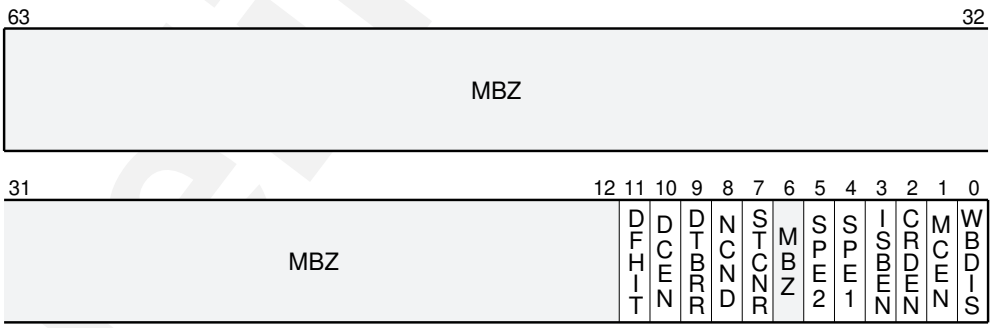
4.3.10 Flush Instruction Cache ASM Register (FLUSH_IC_ASM)

Any write to this pseudo-register invalidates all Icache blocks in which the ASM bit is clear.

4.3.11 Abox Control Register (ABOX_CTL)

Figure 4–22 shows the Abox control register format. Table 4–10 describes the register fields.

Figure 4–22 Abox Control Register (ABOX_CTL)



Preliminary

Table 4–10 Abox Control Register (ABOX_CTL) Bit Description

Bits	Field	Type	Description
0	WBDIS	WO,0	Write Buffer unload Disable. When set, this bit prevents the write buffer from sending write data to the memory controller. It should be set for diagnostics only.
1	MCEN	WO,0	Machine Check Enable. When this bit is set, the Abox generates a machine check when errors (which are not correctable by the hardware) are encountered. When this bit is cleared, uncorrectable errors do not cause a machine check. However, the DC_STAT register is updated and locked when the errors occur.
2	CRDEN	WO,0	Corrected read data interrupt enable. When this bit is set, the Abox generates an interrupt request whenever bus transaction is terminated with a SOFT_ERROR.
3	ISBEN	WO,0	Icache stream buffer enable. When set, this bit enables operation of a single entry Icache stream buffer.
4	SPE1	WO,0	When this bit is set, it enables one-to-one superpage mapping of the D-stream virtual addresses with VA <42:30> = 1FFE to the physical addresses with PA <33:30> = 0. Access is only allowed in kernel mode.
5	SPE2	WO,0	When this bit is set, it enables one-to-one superpage mapping of the D-stream virtual addresses with VA <33:13> directly to physical addresses PA <33:13>, if virtual address bits VA <42:41> = 2. Virtual address bits VA <40:34> are ignored in this translation. Access is only allowed in kernel mode.
7	DTBRR	WO,0	DTB round robin enable. When clear, not-last-used algorithm is used for the DTB. When set, DTB uses round robin replacement.
8	NCND	WO,0	MBZ. When set, Dcache line associated with a read address is not disturbed.

(continued on next page)

Preliminary

Table 4–10 (Cont.) Abox Control Register (ABOX_CTL) Bit Description

Bits	Field	Type	Description
9	STCNR	WO,0	MBZ. For normal operation this bit must be zero. However, when set changes the way LDL_L/LDQ_L and STL_C/STQ_C instructions are handled as follows: The result written in the register identified by the Ra field in STx_C and HW_ST/C instructions will be UNPREDICTABLE. This allows the IBOX to restart the memory reference pipeline when the STx_C is transferred from the write buffer to the memory controller and therefore increases the repetition rate with which STx/C instructions can be processed. LDx_L, STx_C and HW_ST/C instructions invalidate the Dcache line associated with their generated address. These invalidates will not be visible to load or store instructions that issue in the two CPU cycles after the LDL_L/LDQ_L, STL_C/STQ_C or HW_ST/C issues.
10	DCEN	WO,0	Dcache enable. When clear, this bit disables and flushes the Dcache. When set, this bit enables the Dcache.
11	DFHIT	WO,0	Dcache force hit. When set, this bit forces all D-stream references to hit in the Dcache. This bit takes precedence over DC_EN. That is, when DC_FHIT is set and DC_EN is clear all D-stream references hit in the Dcache.

4.3.12 Alternate Processor Mode Register (ALT_MODE)

The ALT_MODE is a write-only register. The AM field specifies the alternate processor mode used by HW_LD and HW_ST instructions that have their ALT bit (bit 14) set. Figure 4–23 shows the alternate processor mode register format and Table 4–11 lists the register modes.

Preliminary

Figure 4–23 Alternate Processor Mode Register (ALT_MODE)

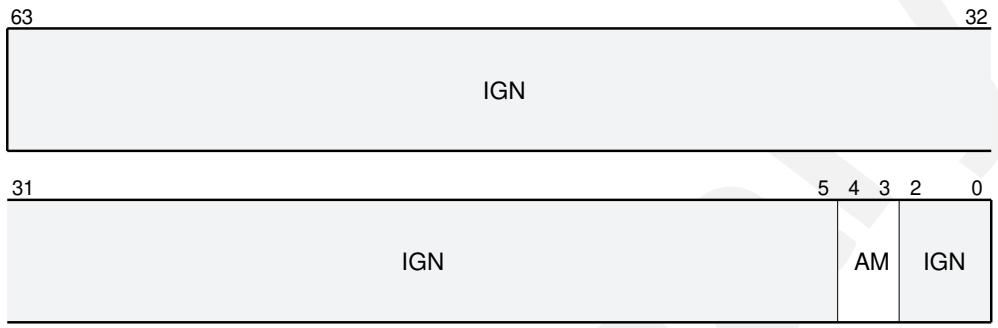


Table 4–11 Alternate Processor Mode Register (ALT_MODE) Bit Description

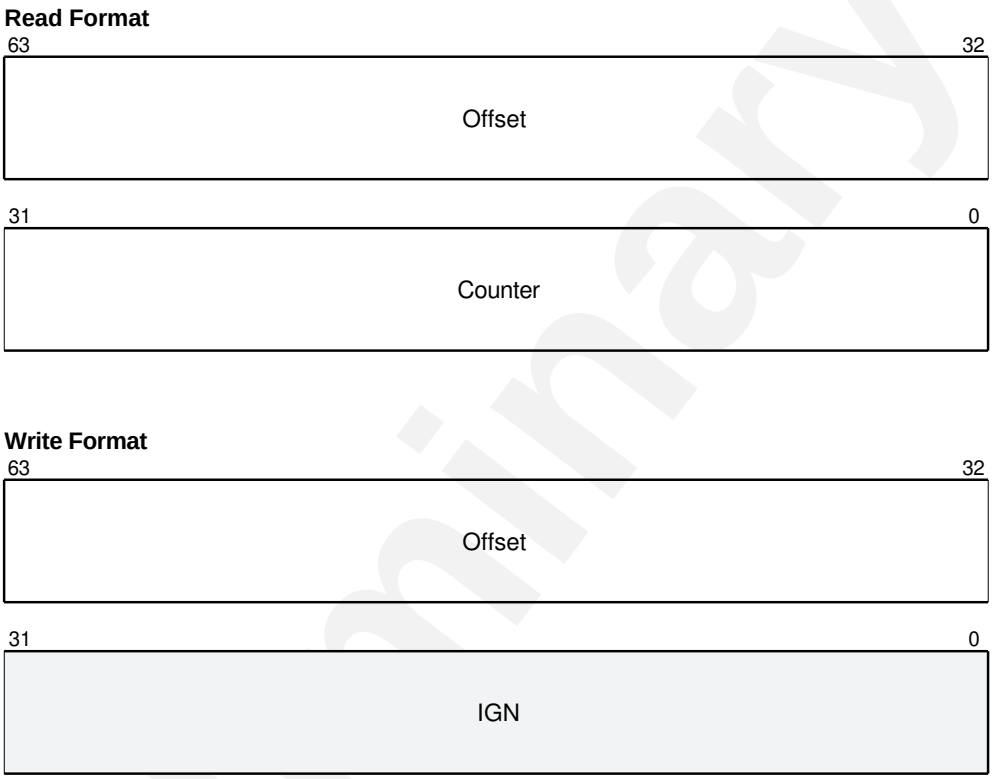
ALT_MODE <4:3>	Mode
00	Kernel
01	Executive
10	Supervisor
11	User

4.3.13 Cycle Counter Register (CC)

The microprocessor supports a cycle counter, as described in the Alpha Architecture Handbook. When enabled, the CC increments once each CPU cycle. The HW_MTPR Rn, CC writes the CC <63:32> with the value held in the Rn <63:32>. The CC <31:0> are not changed. This register is read by the RPCC instruction as defined in the architecture handbook. Figure 4–24 shows the register format (top register) when read by the RPCC instruction and when written (bottom register) by the HW_MTPR Rn, CC instruction.

Preliminary

Figure 4–24 Cycle Counter Register (Read)

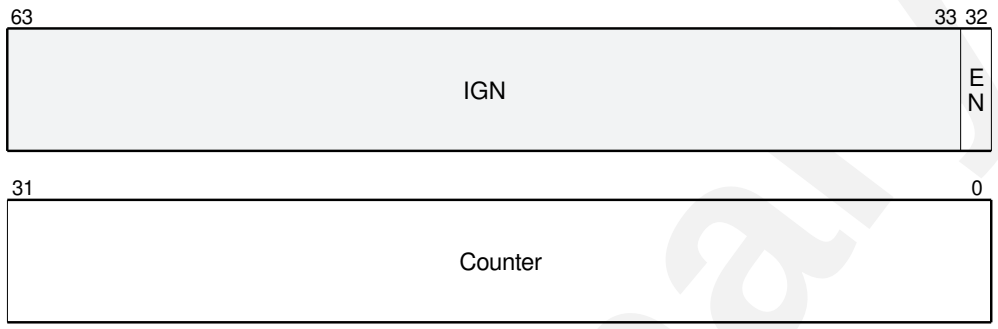


4.3.14 Cycle Counter Control Register (CC_CTL)

The HW_MTPR Rn, CC_CTL writes the CC <31:0> with the value held in Rn <31:0>. The CC register bits <63:32> are not changed. The CC register bits <3:0> must be written with zero. If Rn bit 32 is set, then the counter is enabled, otherwise the counter is disabled. CC_CTL is a write-only register. Figure 4–25 shows the register format when written by the HW_MTPR Rn, CC_CTL instruction.

Preliminary

Figure 4–25 Cycle Counter Control Register



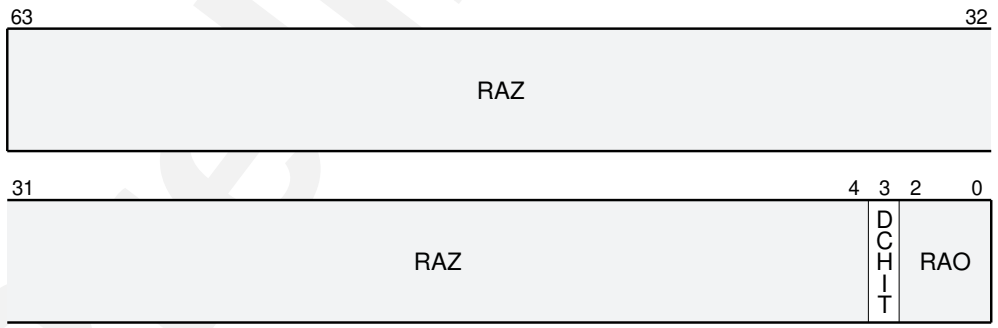
4.4 PAL_TEMP Internal Processor Registers

The CPU chip contains 32 registers that are accessible by way of the HW_MxPR instructions. These registers provide temporary storage for PALcode.

4.4.1 Data Cache Status Register (DC_STAT)

The DC_STAT is a read-only register and is only use by the diagnostics. Figure 4–26 shows the data cache status register format. Table 4–12 describes the register fields.

Figure 4–26 Data Cache Status Register (DC_STAT)



Preliminary

Table 4–12 Data Cache Status Register (DC_STAT) Bit Description

Bits	Field	Type	Description
3	DCHIT	RO	This bit indicates whether the last load or store instruction processed by the Abox hit (DCHIT set) or missed (DCHIT clear) the Dcache. Loads that miss the Dcache can be completed without requiring external reads.

4.5 Internal Processor Registers Reset State

Table 4–13 lists the state of all the internal processor registers (IPRs) immediately following reset. The table also specifies which registers need to be initialized by power-up PALcode.

Table 4–13 Internal Process Register Reset State

IPR	Reset State	Comments
TB_TAG	Undefined	–
ITB_PTE	Undefined	–
ICCSR	Cleared except ASN, PC0, PC1	Floating point disabled, single issue mode, Pipe mode enabled, JSR predictions disabled, branch predictions disabled, branch history table disabled, performance counters reset to zero, Perf Cnt0: Total Issues/2, Perf Cnt1: Dcache Misses, superpage disabled.
ITB_PTE_TEMP	Undefined	–
EXC_ADDR	Undefined	–
SL_RCV	Undefined	–
ITBZAP	NA*	PALcode must do a ITBZAP on reset before writing the ITB (must do HW_MTPR to ITBZAP register).
ITBASM	NA	–
ITBIS	NA	–
PS	Undefined	PALcode must set processor status.
EXC_SUM	Undefined	PALcode must clear exception summary write mask by doing 64 reads.

*NA = not applicable

(continued on next page)

Preliminary

Table 4–13 (Cont.) Internal Process Register Reset State

IPR	Reset State	Comments
PAL_BASE	Cleared	Cleared on reset.
HIRR	NA	–
SIRR	Undefined	PALcode must initialize.
ASTRR	Undefined	PALcode must initialize.
HIER	Undefined	PALcode must initialize.
SIER	Undefined	PALcode must initialize.
ASTER	Undefined	PALcode must initialize.
SL_CLR	Undefined	PALcode must initialize.
SL_XMIT	Undefined	PALcode must initialize. Appears on external pin.
DTB_CTL	Undefined	PALcode must select between SP/LP DTB prior to any TB fill.
DTB_PTE	Undefined	–
DTB_PTE_TEMP	Undefined	–
MM_CSR	Undefined	Unlocked on reset.
VA	Undefined	Unlocked on reset.
DTBZAP	NA	PALcode must do a ITBZAP on reset. See ITBZAP.
DTBASM	NA	–
DTBIS	NA	–
DC_STAT	Undefined	–
ABOX_CTL	Cleared	<11:0> <- ^x0100 Write buffer enabled, machine checks disabled, correctable read interrupts disabled, Icache stream buffer disabled, superpages 1 and 2 disabled, Dcache disabled, and forced hit mode off.
ALT_MODE	Undefined	–
CC	Undefined	Cycle counter is disabled on reset.
CC_CTL	Undefined	–
PAL_TEMP<31:0>	Undefined	–

Preliminary

5
Memory Controller

This chapter gives an overview of the memory controller and describes its registers, interface signals, and cycle timing. It also describes supported memories, memory operations, memory error handling, and graphics operations.

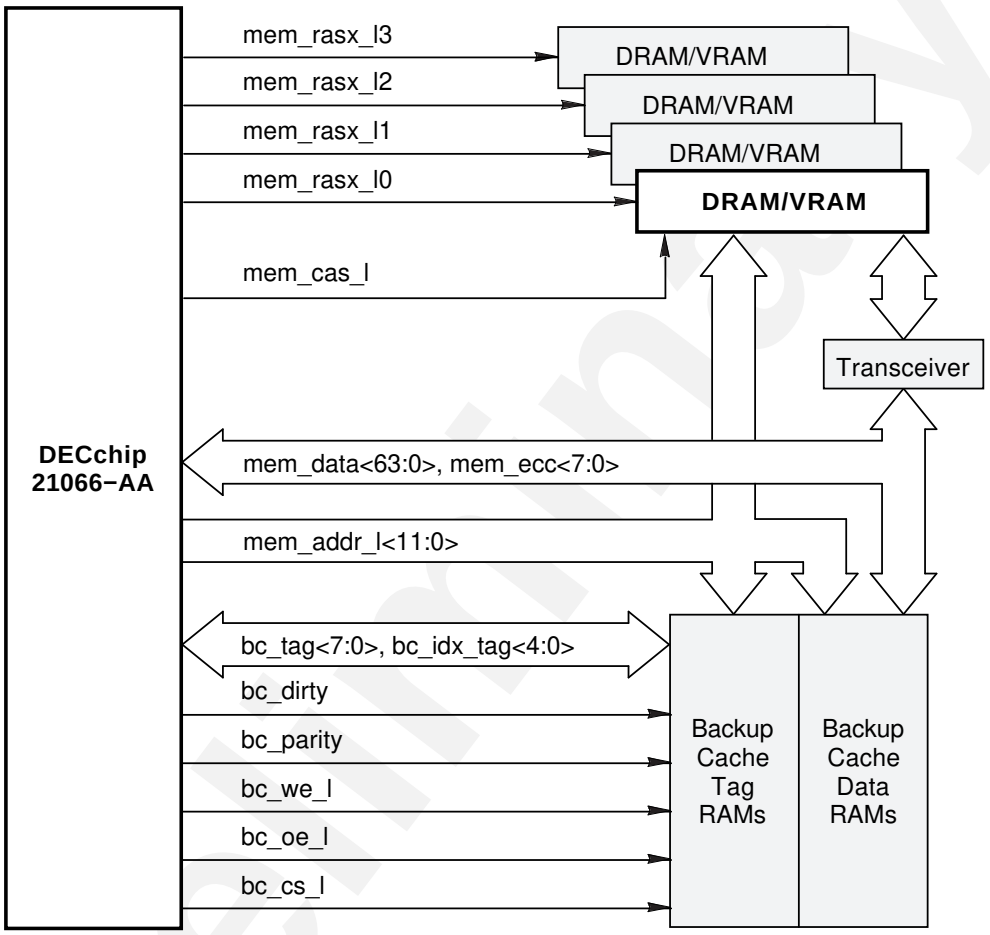
5.1 Introduction

The Memory Controller controls the primary external memory of the system. It has the capability to directly control up to 512 Mbytes of memory organized into four banks of Dynamic Random Access Memory (DRAM) and/or Video Random Access Memory (VRAM), and an optional external backup cache (Bcache) from 64Kbytes to 2Mbytes. Split bank SIMMs are directly supported. Each bank of memory may have a maximum of 16 million addressable locations, each location containing 64 bits. The base address of each bank may be located any where within the lower half of physical memory (restrictions are placed on where the base address of each bank must be. See Section 5.5, Memory Control Registers for complete details). Each of the memory banks and the Bcache are individually configurable via the Bank Configuration, Mask, and Timing Registers and Cache Register.

The Memory Controller has the capability to perform read,write, longword write, byte write (with external logic) and write-per-bit operations to the DRAMs and VRAMs. It has the capability to read and write the Bcache. It incorporates optimizations for fast page-mode access and controls the memory refresh for VRAMs and/or DRAMs. The Memory Controller supports optional ECC (Error Correcting Code) detection and correction. It also supports several graphics operations. Figure 5–1 shows a typical memory system. Note that all banks of memory need not be present in a system.

Preliminary

Figure 5–1 Typical Memory System



Notes: mem_rasx = mem_rasa or mem_rasb.
Signal buffering that may be required due to loading is not shown.
Optional Components

5.1.1 Supported Memory Types

The Memory Controller is capable of controlling Dynamic Random Access Memory (DRAM) and Video Random Access Memory (VRAM) with support for fast page-mode reads and writes and write-per-bit capability. The Memory Controller uses CAS-before-RAS type refreshing of all banks simultaneously.

Preliminary

5.1.2 Supported Memory Sizes

The Memory Controller is capable of controlling up to 512 MBytes of memory. The memory is partitioned into four banks. Each bank is 64 bits wide (excluding ECC bits) and may be any of the following sizes (in quadwords): 128K, 256K, 512K, 1M, 2M, 4M, 8M, or 16M. VRAM bank sizes of 128K, 256K, or 512K are supported. Note that this means that if VRAM is used in one of the banks, then the maximum memory configuration would be 384 MBytes of DRAM and 4 MBytes of VRAM.

Industry-standard single-bank and dual-bank SIMMs are supported. Dual-bank SIMMs are cards with two banks of components, requiring a separate RAS control for each bank. They typically contain components on both sides of the SIMM, for higher component density per SIMM. Dual-bank SIMMs are also commonly called split-bank SIMMs.

5.1.3 Backup Cache

The Memory Controller can support a 64KB, 128KB, 256KB, 512KB, 1MB, or 2MB Bcache. The Bcache is direct map, write-back, quadword block size, and may optionally have data ECC and Tag parity protection. The Tag compare is done on-chip, and all control strobes for the cache RAMs are generated on-chip.

The Tag stores the upper part of the address which is cached at a given location. Which bits of the address are used is a function of the cache size (see Section 5.5.7). The Tag also stores a dirty bit, which indicates that the data stored a that cache location has more recently written data than the corresponding location in memory.

The Memory Controller attempts to minimize overall memory latency in the way it uses the Bcache. In general, the sequence of operation is according to the following rules. For a more detailed description of the operation see Figure 5–3.

If the Bcache is enabled and the address is in cacheable space (bits <33:29> = 0) the Memory Controller will check the cache for a hit.

- For a read hit the cache data will be used and the DRAM will not be read.
- For a read miss, the DRAM will be read. A cache block will be allocated only if the read was from the CPU.
- For a write hit the cache will be updated.
- For a write miss a cache block will be allocated only if the write was from the CPU.

Preliminary

- In either case of allocating a block, if the block being allocated had the tag dirty bit set, the old contents of the block (called the dirty victim block) will be written to DRAM prior to allocating.

If the Bcache is not enabled, or the address is not in cacheable space, the Memory Controller will access the DRAM immediately, omitting the cache access.

Note that the Bcache lookup is independent of page mode operation (that is the Bcache is looked into prior to both page mode and nonpage mode DRAM cycles). See Section 5.2.2 for a description of page mode operation.

5.1.3.1 Backup Cache Initialization

Since the Bcache has no valid bits, it is the responsibility of software to initialize it. The Bcache is initialized by sweeping the entire storage space of the cache with writes. This is done by enabling the Bcache, clearing the parity and ECC enable bits (see Section 5.5.7, Cache Control Register) and then writing enough memory locations to sweep the cache's address range. To avoid nonexistent memory errors, the memory bank registers should be set up to span the largest possible cache address range. At this point the entire Bcache will have valid data and correct tag, parity, and ECC information loaded. The parity and ECC enable bits may then be set.

5.1.4 Programmable Configuration

The above parameters (bank size, ECC, cache), external memory timing parameters, and several error checking parameters are highly programmable via internal Memory Controller registers. See Section 5.5 for details.

5.1.5 Error Correction and Detection

The Memory Controller has the capability to use an 8-bit ECC (Error Correcting Codes) on all memory cycles to banks which support ECC to correct single bit errors, detect double bit errors, and detect 4-bit nibble errors. (4-bit nibble error detection may be very useful when using n ×4 memory parts. The memory data bits must be wired as shown in Table 5–1 for nibble error detection to work properly.) Note that there are cases with more than two bits in error (and are not a full nibble) which appear as either no error, or correctable error. Figure 5–2 shows how the 8 bit code is derived. A 1 in the figure indicates that those data bits are exclusive-ORed to produce the corresponding check bit.

Preliminary

Figure 5–2 ECC Code

Check Bit	D 0	D 63
E0	1101 0001 0001 0011 1110 1101 1101 1100 0011 0010 0010 1101 1110 1010 1110 0000	
E1	1010 0010 0010 0100 1011 1010 1010 1011 0100 0100 0100 1011 1011 1101 1101 1000	
E2	0111 1100 1100 1000 0101 0111 0111 0111 1000 1001 1001 0110 0101 0111 0011 0000	
E3*	0100 1111 1111 0001 0010 0001 0001 0001 0001 0001 0001 0001 1111 0001 1111 1100	
E4	1111 0101 1001 0111 0001 0100 0100 1000 0111 1101 1101 0010 0110 1000 0110 0010	
E5	1111 1010 0110 1101 0100 1000 1000 0010 1101 1011 1011 0100 1001 0100 1001 0001	
E6	1011 1111 0000 0100 1000 1101 0010 0100 1011 1000 0111 0111 1111 1101 0000 1111	
E7*	0000 1111 0000 1111 0000 1111 0000 0000 0000 1111 0000 1111 0000 1111 1111 0111	

Notes:
Exculsive OR is done on inverse of data.
* E3 and E7 are stored active low.

During memory read cycles, the stored ECC is exculsive-ORed with the expected ECC to produce an 8 bit Syndrome. If the two codes match, then no error was detected.If the codes differ, the syndrome is checked to determine if the error is correctable.

Table 5–1 Nibble Correction Bit Grouping

D0	D1	D2	D3
D4	D5	D6	D7
D8	D9	D10	D11
D12	D13	D14	D15
D16	D17	D18	D19
D20	D21	D22	D23
D24	D25	D26	D27
D28	D29	D30	D31
D32	D33	D34	D35
D36	D37	D38	D39
D40	D41	D42	D43
D44	D45	D46	D47
D48	D49	D50	D51
D52	D53	D54	D55
D56	D57	D58	E0
D59	D60	E1	E2
D61	D62	D63	E6

(continued on next page)

Preliminary

Table 5–1 (Cont.) Nibble Correction Bit Grouping

E3	E4	E5	E7

5.1.6 Graphics Features

The Memory Controller has the capability of performing some simple graphics oriented data manipulations. The CPU is responsible for generating the address of all graphics operations. The graphics features are:

- Dumb Frame Buffer Operation
- Transparent Stipple Operation
- Write-Per-Bit Plane Masking
- Byte Writes (with external gating)
- Full and Split Shift Register Loads

These operations are programmable via the Graphics Control Register described in Section 5.5, and are only used on writes to addresses in Graphics address region (which is also noncacheable).

5.2 Memory Controller Operation

The Memory Controller decodes all addresses from CPU and PCI DMA device reads and writes. If the address is in memory space (see Table 2–5) then the Memory Controller will follow the sequence shown in Figure 5–3.

5.2.1 Masked Writes

A masked write is a write that modifies less than the full quadword of data at an address. They are the result of longword store instructions that did not get merged in the CPU Write Buffer, or of PCI DMA writes (which can specify writes on a byte basis), or of writes to graphics address space (Table 2–5).

Since ECC covers the full quadword, masked writes to memory which is protected by ECC must be handled by read-modify-writes, in which the Memory Controller reads the memory data, merges it internally with the write data, and then writes the result back to memory. Banks of DRAM or VRAM which are not covered by ECC may optionally have write-per-bit memories (normally VRAMs have this) or, with some minimal external logic, be made byte writable. Both of these are enabled by bits in the Bank Configuration Register (BCR, Section 5.5.1).

Preliminary

If the address is in graphics space and the addressed bank BCR has write-per-bit enabled the Memory Controller drives the write-per-bit mask on the data lines during the row address period. ECC for the write data is driven during this operation, although it is not used by the memories. If the addressed bank BCR has byte write enabled, the Memory Controller drives the byte write mask on the **mem_ecc<7:0>** pins during the column address period. External logic is required to gate CAS at the DRAMs/VRAMs. Note that some masked writes to byte writeable banks will be done as read-modify-writes. For a masked write to skip the read portion, either of the following conditions must be met:

- The address is in noncacheable space.
- The address is in graphics space and the VGR (Section 5.5.8) byte write enable bit is set.

The basic difference between write-per-bit and byte write is that write-per-bit can modify arbitrary bits, while byte write can only modify memory on a byte resolution basis. This means that byte write mode will normally be used with pixel widths that are integral numbers of bytes. Note that write-per-bit and byte write may be used simultaneously.

The amount of time required to perform any memory operation is determined by the values programmed into the Cache and Memory Timing Registers, whether or not the Bcache hits, and whether or not a required DRAM operation is done in page mode.

5.2.2 Page Mode

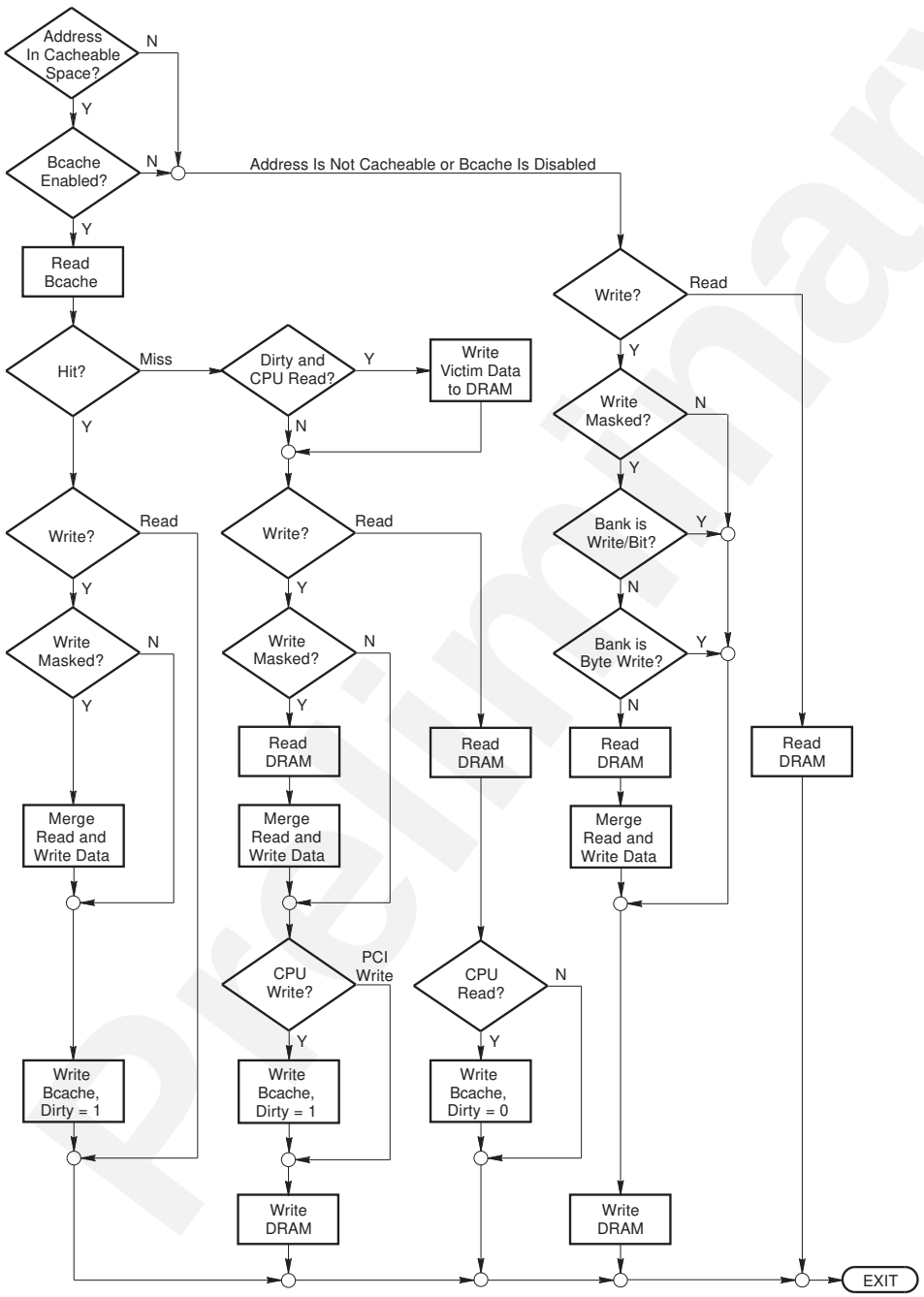
Page mode cycles are cycles that do not need to deassert RAS to latch a new row address into the DRAMs. The cycles can be any combination of reads and writes. The Memory Controller will do a page mode cycle when RAS is already asserted for a bank, and a read or write request is made with the same page as when RAS was asserted. The determination of same page is a function of DRAM column size and is made by comparing address bits as shown in Table 5–4.

5.2.3 Register Access

The Memory Controller contains a number of registers described in Section 5.5. They can be read and written by the CPU. They may not be read or written by I/O devices. Refer to Table 5–2 for the names and addresses.

Preliminary

Figure 5–3 Memory Controller Operation



Preliminary

5.3 Memory Error Conditions

The Memory Controller is responsible for checking and logging of various memory error conditions. Associated with each possible memory error is a bit in the Error Status Register. At the occurrence of any error, the proper bit in the Error Status Register (Section 5.5.5) is set, and the address of the location with the error is stored in the Error Address Register (Section 5.5.6).

An interrupt is sent to the CPU when any error condition is detected. It is the responsibility of software to clear any error condition bits and return the memory subsystem to proper operation. The occurence of another error prior to the CPU clearing the first error will set a multiple error bit, but no information about the error type will be saved.

The following sections describe the errors and action taken during each error type.

5.3.1 Correctable Read Error

A correctable read error indicates that a single bit of data was corrected during a memory read or read part of a read-modify-write operation done during a masked write. The Memory Controller corrects the data before sending it to the initiator of the read (either CPU or PCI device).

Detection of this error will set the MCE (Memory Correctable Error) bit in the Error Status Register. An interrupt is sent to the CPU to indicate that an error was detected. It is the responsibility of soft ware to write the correct data to the address which had the single bit error. This may be done with a LDQ_L and STQ_C sequence.

5.3.2 Uncorrectable Read Error

An uncorrectable read error occurs whenever the ECC logic detects more than one data bit in error (see Section 5.1.5). The initiator of the read (either CPU or PCI device) is informed that an error has occurred. How each of those respond is described in the appropriate sections of this spec.

Detection of this error will set the MUE (Memory Uncorrectable Error) bit in the Error Status Register. An interrupt is sent to the CPU to indicate that an error was detected. It is the responsibility of software to re-establish the validity of the faulting memory location, if possible.

Preliminary

5.3.3 Uncorrectable Write Error

An uncorrectable write error may occur when a read-modify-write operation is done during a masked write. The error is detected during the read portion of the operation. Since the write data can not be correctly merged with the (uncorrectable) read data, the memory write operation is aborted.

Detection of this error will set the MWE (Memory Write Error) bit in the Error Status Register. An interrupt is sent to the CPU to indicate that an error was detected. It is the responsibility of software to re-establish the validity of the faulting memory location, if possible.

Aborting the memory write operation preserves the uncorrectable read data in memory. A subsequent read of the location will return an uncorrectable read status. It is left to the discretion of the programmer as to whether a failed write operation is considered a fatal problem.

5.3.4 Bcache Tag Parity Error

Each time the Memory Controller accesses the Bcache the Tag, Dirty Bit and Tag Parity bit will be checked for odd parity (the parity is calculated on the **bc_tag<7:0>** (Section 5.6.11) and **bc_tag_idx<4:0>** (Section 5.6.15) pins). If the Tag parity is enabled and the stored parity is incorrect a Tag Parity Error occurs.

If the Tag Parity Error occured on a read the initiator of the read (either CPU or PCI device) is in formed that an error has occurred. How each of those respond is described in the appropriate sections of this spec.

Detection of this error will set the CTE (Bcache Tag Parity Error) bit in the Error Status Register. An interrupt is sent to the CPU to indicate that an error was detected. It is the responsibility of software to re-establish the Bcache's coherency, if possible.

5.3.5 Nonexistent Memory Address Error

A nonexistent Memory Address Error occurs when a read or write is attempted to DRAM at an address which is not programmed in the Bank Address and Mask Registers.

If the nonexistent Memory Address Error occured on a read the initiator of the read (either CPU or PCI device) is informed that an error has occurred. How each of those respond is described in the appropriate sections of this spec.

Detection of this error will set the NXM (nonexistent Memory Error) bit in the Error Status Register. An interrupt is sent to the CPU to indicate that an error was detected. Note that the Memory Controller allocates the Bcache on writes. A write to a nonexistent address will allocate the cache and set

Preliminary

the NXM bit at the same time. It will also detect the same NXM error at a later time, when the cache location is reallocated, and the dirty data must be written to DRAM.

5.4 Graphics Operations

The Memory Controller provides some simple graphics hardware features that improve frame buffer performance.

The Memory Graphics Functions are used on memory writes when address bits <33:29> select an address in Graphics Space. The graphics function performed depends on the contents of the Video and Graphics Control Register (VGR, Section 5.5.8). In all other address regions the graphics functions are not enabled.

To facilitate the use of VRAMs, the Memory Controller supports both full and split shift register loads. External monitor timing logic must be used to signal the Memory Controller when VRAM shift register loads need to be done. An internal linear address generator (Video Display Pointer – Section 5.5.8.1) is provided to keep track of the video refresh address. Reads ignore the settings of the VGR; graphics operations are only done on writes.

5.4.1 Simple Frame Buffer Mode

In the Simple Frame Buffer Mode, the Memory Controller can act exactly like an interface to a dumb frame buffer. The Plane Mask Register (Section 5.5.9) is used to mask writing some of the bits in memory. This is done using either write-per-bit, byte write, or read-modify write; see Section 5.2.1. A bit will only be written if the corresponding Plane Mask Register bit is a '1'. For byte write mode the byte mask driven out on the **mem_ecc<7:0>** pins for each byte is determined by the instruction (STL or STQ) and longword address.

5.4.2 Transparent Stipple Mode

In Transparent Stipple Mode, CPU write cycles can conditionally substitute the foreground data value for the frame buffer data.

The CPU write address specifies the address which will be written and the write data and Plane Mask Register are used to mask writing some of the bits in memory. The content of the Foreground Register (Section 5.5.10) is used as write data. This is done using either byte write, write-per-bit or read-modify write; see Section 5.2.1. A bit will only be written if the corresponding write data and Plane Mask Register bit are both '1'. For byte write mode the byte mask driven out on the **mem_ecc<7:0>** pins for each byte is determined by the instruction (STL or STQ) and longword address, and the lsb of the corresponding byte write data.

Preliminary

5.5 Registers

This section contains the definitions for the Registers used to configure the Memory Controller. All registers are 64 bits wide, aligned on quadword addresses. However, in all cases except the Error Status and Plane Mask Registers bits <63:32> are unused so are not shown in the figures.

Before memory operations may be initiated, at least one banks' registers must be initialized with the proper memory parameters.

The register addresses are in Table 5–2. The addresses shown are the preferred address to use when accessing Memory Controller registers. In actuality only bits <7:3> of the address are decoded within the register range, so each register may be accessed at multiple addresses. However, future versions of the Memory Controller reserve the right to use those addresses for different purposes.

Table 5–2 Memory Controller Registers

Mnemonic	Register	Address (Hexadecimal)
BCR0	Bank Configuration 0	1 2000 0000
BCR1	Bank Configuration 1	1 2000 0008
BCR2	Bank Configuration 2	1 2000 0010
BCR3	Bank Configuration 3	1 2000 0018
BMR0	Bank Mask 0	1 2000 0020
BMR1	Bank Mask 1	1 2000 0028
BMR2	Bank Mask 2	1 2000 0030
BMR3	Bank Mask 3	1 2000 0038
BTR0	Bank Timing 0	1 2000 0040
BTR1	Bank Timing 1	1 2000 0048
BTR2	Bank Timing 2	1 2000 0050
BTR3	Bank Timing 3	1 2000 0058
GTR	Global Timing	1 2000 0060
ESR	Error Status	1 2000 0068
EAR	Error Address	1 2000 0070
CAR	Cache Control	1 2000 0078
VGR	Video and Graphics Control	1 2000 0080
PLM	Plane Mask	1 2000 0088
FOR	Foreground	1 2000 0090

Preliminary

5.5.1 Bank Configuration Registers (BCR3–BCR0)

The Bank Configuration Registers are used to program the Base address, error mode, write mode, and Row address manipulation for each bank of memory. Note that each bank has its own BCR. All defined bits of these registers are write only. At RESET, all bits are undefined except for BASE ADDRESS VALID (BAV) which is cleared at RESET.

Figure 5–4 Bank Configuration Register (BCR3–BCR0)

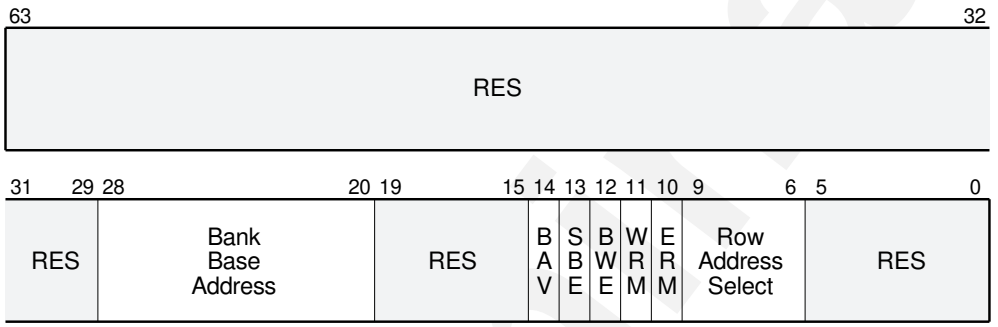


Table 5–3 Bank Configuration Register (BCR3–BCR0) Bit Description

Bits	Field	Type	Description
5:0	RES	RAX/IGN	Reserved
9:6	Row Address Select	WO	These bits specify how the memory address should be multiplexed for use by this bank of memory. Table 5–4 lists the assignment of these bits for the number of row and column addresses and Figure 5–5 shows how the address bits are multiplexed. Column addresses always consist of address bits <14:3>.
10	ERM	WO	Error Mode. This bit selects whether or not ECC should be checked/generated for each access to this bank. Note: Memories which use the Write-per-bit (bit 6 of this register) or Byte Write capability (bit 7 of this register) should not be programmed to use ECC.

(continued on next page)

Preliminary

Table 5–3 (Cont.) Bank Configuration Register (BCR3–BCR0) Bit Description

Bits	Field	Type	Description
11	WRM	WO	Write Mode. This bit selects whether the bank has write-per-bit capability or not. Note: Memories which use the write per-bit capability are not protected with ECC.
12	BWE	WO	Byte Write Enable. This bit enables external byte masked writes for graphics operations. The byte mask is determined by the graphics mode (Section 5.5.8), Foreground Register (Section 5.5.10), and Plane Mask Register (Section 5.5.9). Note that to use byte masking ECC must be disabled for the bank (bit 5 of this register). The byte mask is driven on the **mem_ ecc<7:0>** (Section 5.6.2) pins and external logic is required to gate the RAM write signals for each byte.
13	SBE	WO	Split Bank Enable. This bit enables split bank operation for the bank. This controls RAS selection as shown in Table 5–5 using the address bit as shown in Table 5–7.
14	BAV	WO	Base Address Valid. This bit indicates if the base address is valid. It should be set when the base address is written to enable the addressing of the associated memory bank. At reset, this bit is cleared. Before setting this bit after reset, the corresponding Bank's BMR (Section 5.5.2) must be written with valid information.
19:15	RES	RAX/ IGN	Reserved
28:20	Bank Base Address	WO	This field specifies the base or starting physical address of the associated bank of memory.Each bank of memory must begin on a naturally aligned boundary. Naturally aligned means that for a bank with 2n addresses, the n least significant bits must be zero. The Bank Base Address is compared to Physical Address bits <28:20> and masked by the Bank Base Address Mask (Section 5.5.2).
63:29	RES	RAX/ IGN	Reserved

Preliminary

Table 5–4 Row Address Select Encodings

Code	Column Address	Row Address	Bank* Size	Page† Size	Address Bits Used For Page Compare
0000	8	9	128K	256	28:11
		10	256K		
		11	512K		
		12	1M		
0001	9	9	256K	512	28:12
		10	512K		
		11	1M		
		12	2M		
0011	10	10	1M	1024	28:13
		11	2M		
		12	4M		
0111	11	11	4M	2048	28:14
		12	8M		
1111	12	12	16M	4096	28:15

* Number of quadwords = $2^{row+col}$
† Number of quadwords = 2^{col}

Table 5–5 RAS Selection

Split Bank Enable	Address Bit*	mem_rasa_l<3:0>	mem_rasb_l<3:0>
0	X	Y	N
1	0	Y	N
1	1	N	Y

*From Table 5–7.

Preliminary

Figure 5–5 Row Address Bit Assignments

Number of Row Addresses:	9	10	12	9	10	12	10	11	12	11	12	12
Column Addresses:	8	8	8	9	9	9	10	10	10	11	11	12
mem_adr11	**22**	**22**	22	**22**	**22**	22	**24**	**24**	24	**25**	25	25
mem_adr10	**21**	**21**	21	**23**	**23**	23	**23**	23	23	23	23	23
mem_adr9	**20**	20	20	**21**	21	21	21	21	21	21	21	21
mem_adr8	11	11	11	20	20	20	20	20	20	20	20	20
mem_adr7	12	12	12	12	12	12	22	22	22	22	22	22
mem_adr6	13	13	13	13	13	13	13	13	13	24	24	24
mem_adr5	14	14	14	14	14	14	14	14	14	14	14	26
mem_adr4	19	19	19	19	19	19	19	19	19	19	19	19
mem_adr3	18	18	18	18	18	18	18	18	18	18	18	18
mem_adr2	17	17	17	17	17	17	17	17	17	17	17	17
mem_adr1	16	16	16	16	16	16	16	16	16	16	16	16
mem_adr0	15	15	15	15	15	15	15	15	15	15	15	15

Notes:
Addresses in **small boldface** are driven on **mem_adr** but are not needed by memory parts.
Column addresses always consist of address bits 14:3.

5.5.2 Bank Address Mask Registers (BMR0-BMR3)

The Bank Address Mask Register controls what bits of the base address should not be checked for an address match. Note that each bank has its own BMR. These registers are undefined at reset. This register must be written with valid information before setting the corresponding Bank's BCR (Section 5.5.1) Valid bit.

Preliminary

Figure 5–6 Bank Address Mask Register (BMR3–BMR0)

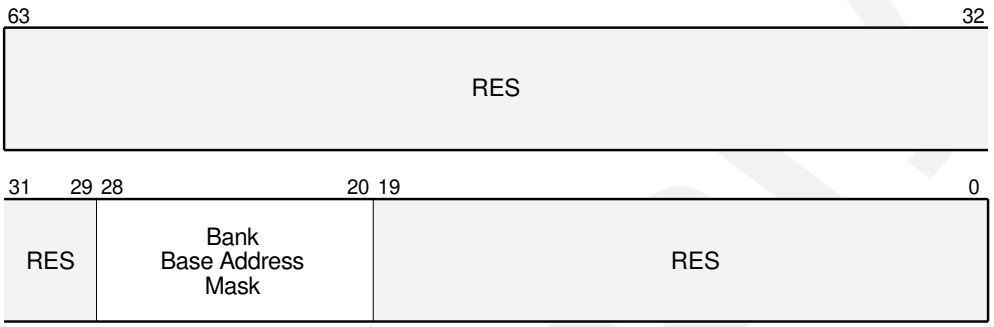


Table 5–6 Bank Address Mask Register (BMR3–BMR0) Bit Description

Bits	Field	Type	Description
19:0	RES	IGN	Reserved
28:20	Bank Base Address Mask	WO	This field specifies the mask to be used when matching the memory address to the configured memory address range. A '1' in a bit position means mask (ie: do not check) the corresponding base address bit. The mask corresponds to physical address bits <28:20>. Figure 5–7 shows how to set the Bank Addr Mask for various size banks of memory.
63:29	RES	IGN	Reserved

Figure 5–7 Bank Size Mask Setting

Memory Bank Size	Physical Address and Mask Register Bits 28	27	26	25	24	23	22	21	20
1MB	0	0	0	0	0	0	0	0	0
2MB	0	0	0	0	0	0	0	0	1
4MB	0	0	0	0	0	0	0	1	1
8MB	0	0	0	0	0	0	1	1	1
16MB	0	0	0	0	0	1	1	1	1
32MB	0	0	0	0	1	1	1	1	1
64MB	0	0	0	1	1	1	1	1	1
128MB	0	0	1	1	1	1	1	1	1

Preliminary

Note that since bits <32:29> are not compared against in the physical address compare it will be possible to address memory at multiple addresses. It is the responsibility of software to insure that this does not cause the system to malfunction.

The mask is also used with the Split Bank bit in BCR to determine which address bit to use in asserting **mem_rasa_l*n*** or **mem_rasb_l*n*** per bank according to Table 5–7.

Table 5–7 Split Bank Address Select

Bank Base Address Mask BMR <27:20>	Address Bit* That Selects mem_rasa_l*n* or mem_rasb_l*n*
01111111	26
00111111	25
00011111	24
00001111	23
00000111	22
00000011	21

*Address bit value 0 selects **mem_rasa_l*n*** and value 1 selects **mem_rasb_l*n***, where *n* = 3, 2, 1, or 0.

5.5.3 Bank Timing Registers (BTR0-BTR3)

The timing parameters for each of the four banks of memory are independently programmable, in cycles, via the Bank Timing Registers. For each of the timing parameters for the bank the delay is the number programmed plus one cycles except where stated otherwise.

All bits of this register are read/write. At RESET, all bits are undefined.

Table 5–21 and timing diagrams in Section 5.7 identify the timing parameters.

Preliminary

Figure 5–8 Bank Timing Register (BTR3–BTR0)

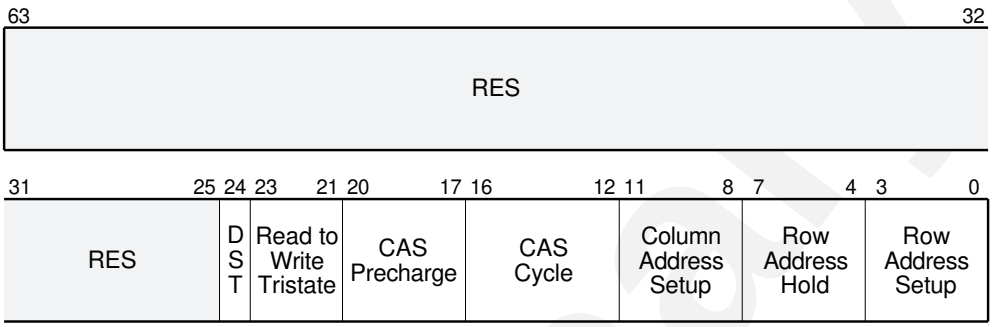


Table 5–8 Bank Timing Register (BTR3–BTR0) Bit Description

Bits	Field	Type	Description
3:0	Row Address Setup	WO	Number of cycles between when row address is driven on **mem_addr<11:0>** and when **mem_ras_l** is asserted. Timing diagram parameter T_{01}; DRAM parameter t_{RAS}.
7:4	Row Address Hold	WO	Number of cycles between when **mem_ras_l** is asserted and **mem_addr<11:0>** switches from row address to column address. Timing diagram parameter T_{12}; DRAM parameter t_{RAH}.
11:8	Column Address Setup	WO	Number of cycles (not number of cycles plus one) between when column address, write data (during memory writes) and write enable are driven and when **mem_cas_l** is asserted. Timing diagram parameter T_{23}; DRAM parameter t_{ASC}.
16:12	CAS Cycle	WO	Number of cycles that **mem_cas_l** is asserted. Timing diagram parameter T_{34}; DRAM parameter t_{CAC}.
20:17	CAS Precharge	WO	Number of cycles (not number of cycles plus one) that **mem_cas_l** is deasserted between page mode cycles. Also represents the setup time before **mem_cas_l**assertion of the column address, write data (during memory writes) and write enable. Timing diagram parameter T_{45}; DRAM parameter t_{CP}.
23:21	Read to Write TriState	WO	Number of cycles required to tristate the memory data bus when switching from a read to write operation. DRAM timing parameter t_{OFF}.

(continued on next page)

Preliminary

Table 5–8 (Cont.) Bank Timing Register (BTR3–BTR0) Bit Description

Bits	Field	Type	Description
24	DST	WO	Data Setup. Determines the minimum number of cycles of data setup prior to **mem_cas_l** assertion on writes (0 = 1 cycle; 1 = two cycles). DRAM timing paramater t $_{DS}$.
63:25	RES	RAX/ IGN	Reserved

5.5.4 Global Timing Register (GTR)

The parameters in this register select the timing parameters for all four banks of memory. For each of the timing parameters the delay is the number programmed plus one except where stated otherwise.

Figure 5–9 Global Timing Register (GTR)

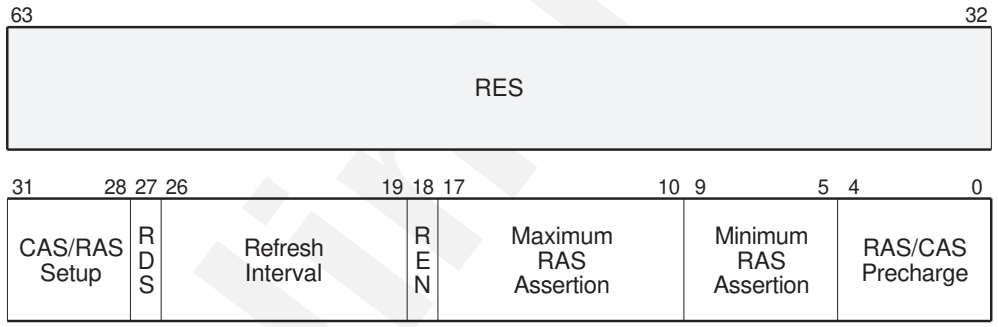


Table 5–9 Global Timing Register (GTR) Bit Description

Bits	Field	Type	Description
4:0	RAS/CAS Precharge	RW	Number of cycles to precharge RAS and CAS at the end of a memory cycle. When the precharge time has elapsed, a new memory cycle may begin. Timing diagram parameter T $_{RASprech}$; DRAM parameter t$_{RP}$.
9:5	Minimum RAS Assertion	RW	Number of cycles that **mem_ras_l** is asserted during a refresh cycle. DRAM parameter t$_{RAS}$. The number of cycles is this value (not this value plus one).

(continued on next page)

Preliminary

Table 5–9 (Cont.) Global Timing Register (GTR) Bit Description

Bits	Field	Type	Description
17:10	Maximum RAS Assertion	RW	Maximum number of cycles (prescaled by 128) that **mem_ras_l** may be asserted. **mem_ras_l** stays asserted for page mode cycles, which, depending on program and I/O behavior, may last for a long time. At the assertion of **mem_ras_l**, this value is loaded into a down counter and decremented once ever 128 cycles. If the count reaches zero and **mem_ras_l** is still asserted, the Memory Controller will complete the current memory cycle and then deassert **mem_ras_l**. DRAM parameter t_{RAS}.
18	REN	RW	Refresh Enable. Enables refresh operations for all of the banks. This bit is cleared at reset. When it is clear the refresh counter is initialized to the value in Refresh Interval. Note that Refresh Interval and Refresh Divide Select are not initialized at reset. Any time they are changed Refresh Enable should be deasserted to allow them to load.
26:19	Refresh Interval	RW	Number of cycles (prescaled by 64 or 512 as selected by Refresh Divide Select) between requests to perform a refresh cycle. The request is generated internally, and the refresh occurs at the next available free cycle (ie after any memory cycles in progress complete). The refresh counter reloads this value when it requests the refresh cycle. This value does not have one added to it, so a value of zero will continually refresh the memory.
27	RDS	RW	Refresh Divide Select. This bit selects the prescale divide for the refresh interval. 0 = divide by 64; 1 = divide by 512.
31:28	CAS/RAS Setup	RW	Number of cycles between assertion of **mem_cas_l** and assertion of **mem_ras_l** on refresh cycles. The number of cycles is this value plus two.
63:32	RES	RAX/ IGN	Reserved

5.5.5 Error Status Register (ESR)

The Error Status Register controls the error detection/correction functions of the Memory Controller and holds the error status when an error occurs.

ECC Bits are read only. Write Wrong ECC is read/write. The error status flags (CEE, UEE, CTE, MSE, MHE) may be read and are cleared when a 1 is written to them. Writing a 0 to those bits has no effect. ESR is unaffected by reset.

Preliminary

Figure 5–10 Error Status Register (ESR)

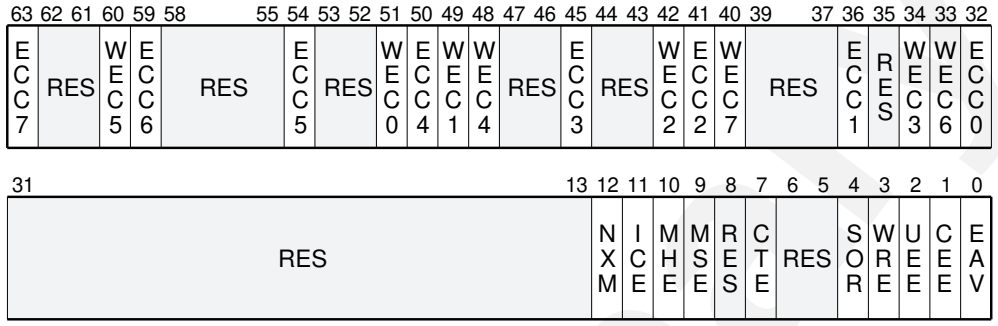


Table 5–10 Error Status Register (ESR) Bit Description

Bits	Field	Type	Description
0	EAV	RO	Error Address Valid. This bit is set to indicate that the address in the Error Address Register is a valid error address. It is the logical OR of bits <12,8:7,2:1>. While it is asserted the Memory Controller asserts an interrupt to the CPU, and bits <12,8:7,4:1> of ESR are frozen, as is the Error Address Register (Section 5.5.6).
1	CEE	R/W1C	Correctable Error. This bit is set when a correctable ECC error is occurs during a read or read-modify-write.
2	UEE	R/W1C	Uncorrectable Error. This bit is set when an uncorrectable ECC error is occurs during a read or read-modify-write.
3	WRE	RO	Write Error. This bit indicates whether an error occured during a read or write access (0 = read, 1 = write). It is updated on each cycle and frozen when the EAV bit of this register is set.
4	SOR	RO	Error Source. This bit indicates whether an error occured while accessing Bcache or Memory (0 = cache, 1 = memory). It is updated on each cycle and frozen when the EAV bit of this register is set.
6:5	RES	RAX/ IGN	Reserved
7	CTE	R/W1C	Bcache Parity Error. This bit is set to indicate that a Tag parity error was detected during a Bcache cycle.

(continued on next page)

Preliminary

Table 5–10 (Cont.) Error Status Register (ESR) Bit Description

Bits	Field	Type	Description
8	RES	RAX/ IGN	Reserved
9	MSE	R/W1C	Multiple Soft Errors. This bit indicates that an additional soft (correctable) error occured after EAV was set. Note that the status about the secondary error is not logged.
10	MHE	R/W1C	Multiple Hard Errors. This bit indicates that an additional hard (uncorrectable) error occured after EAV was set. Note that the status about the secondary error is not logged.
11	ICE	RW	Ignore Correctable Errors. This bit when set prevents the logging of correctable (soft) errors. Note that the data is still corrected before being used; the CEE and EAV bits are not set, and EAR (Section 5.5.6) is not frozen due to a soft error.
12	NXM	R/W1C	Nonexistent Memory. This bit is set when a read or write is attempted to DRAM at an address which is not programmed in the Bank Address and Mask Registers.
31:13	RES	RAX/ IGN	Reserved
32	ECC0	RO	This field (<32,36,41,45,54,59,63>) stores the the ECC read from the **mem_ecc<7:0>** pins. This field is frozen when the ESR Register EAV bit is set.
33	WEC6	RW	Write Wrong ECC. These bits (<33,34,40,42,48,49,51,60>) are used during test of the ECC detection and correction logic. When set to a one, the corresponding ECC Check bit will be written to memory or cache incorrectly. For normal operation, the value written into these bits should be zero.
34	WEC3	RO	See bit 33 description.
35	RES	RAX/ IGN	Reserved
36	ECC1	RO	See bit 32 description.
39:37	RES	RAX/ IGN	Reserved
40	WEC7	RO	See bit 33 description.

(continued on next page)

Preliminary

Table 5–10 (Cont.) Error Status Register (ESR) Bit Description

Bits	Field	Type	Description
41	ECC2	RO	See bit 32 description.
42	WEC2	RO	See bit 33 description.
44:43	RES	RAX/ IGN	Reserved
45	ECC3	RO	See bit 32 description.
47:46	RES	RAX/ IGN	Reserved
48	WEC4	RO	See bit 33 description.
49	WEC1	RO	See bit 33 description.
50	ECC4	RO	See bit 32 description.
51	WEC0	RO	See bit 33 description.
53:52	RES	RAX/ IGN	Reserved
54	ECC5	RO	See bit 32 description.
58:55	RES	RAX/ IGN	Reserved
59	ECC6	RO	See bit 32 description.
60	WEC5	RO	See bit 33 description.
62:61	RES	RAX/ IGN	Reserved
63	ECC7	RO	See bit 32 description.

5.5.6 Error Address Register (EAR)

The Error Address Register holds the address of a cycle which has produced an error. It loads the address each time a memory cycle is done and and freezes when the Error Status Register (Section 5.5.5) EAV is set.

This register is read only and is undefined at reset.

Preliminary

Figure 5–11 Error Address Register (EAR)

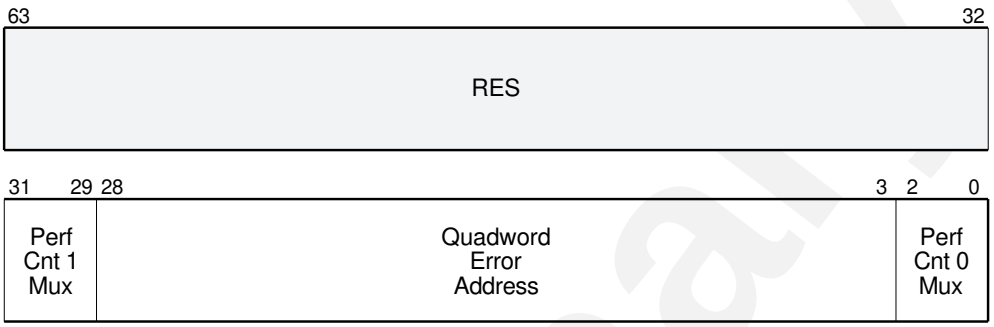


Table 5–11 Error Address Register (ESR) Bit Description

Bits	Field	Type	Description
2:0	Perf Cnt Mux	WO	Performance Count Multiplexer. These bits select one of eight events to be counted for performance evaluation. The events are listed in Table 5–12.
28:3	Quadword Error Address	RO	This field identifies the quadword address (bits <28:3>) where an error occurred during a memory operation. The first address to have an error is logged in this register and held until the EAV bit of ESR (Section 5.5.5) is cleared.
31:29	Perf Cnt Mux	WO	Performance Count Multiplexer. These bits select one of eight events to be counted for performance evaluation. The events are listed in Table 5–12.
63:32	RES	RAX/ IGN	Reserved

Table 5–12 Performance Count Events

Event 0 Select	Event	Event 1 Select	Event
0	Number of reads to Bcache, either from CPU or DMA	0	Number of events from event 0 that are also from CPU and Bcache hits
1	Number of writes to Bcache, either from CPU or DMA	1	Number of events from event 0 that are also from CPU and Bcache misses and clean

(continued on next page)

Preliminary

Table 5–12 (Cont.) Performance Count Events

Event 0 Select	Event	Event 1 Select	Event
2	Number of reads to DRAM, either from CPU or DMA	2	Number of events from event 0 that are also from CPU and Bcache misses and dirty
3	Number of writes to DRAM, either from CPU or DMA	3	Number of events from event 0 that are also from DMA and Bcache hits
4	Number of DRAM accesses that do page mode cycles	4	Number of events from event 0 that are also from DMA and Bcache misses
5	Number of DRAM accesses that miss page mode *	5	Number of CPU writes that write less than a full quadword
6	Number of reads to graphics address space	6	Number of DMA writes that write less than a full quadword
7	Number of writes to graphics address space	7	Number of chip cycles that the memory controller is idle†

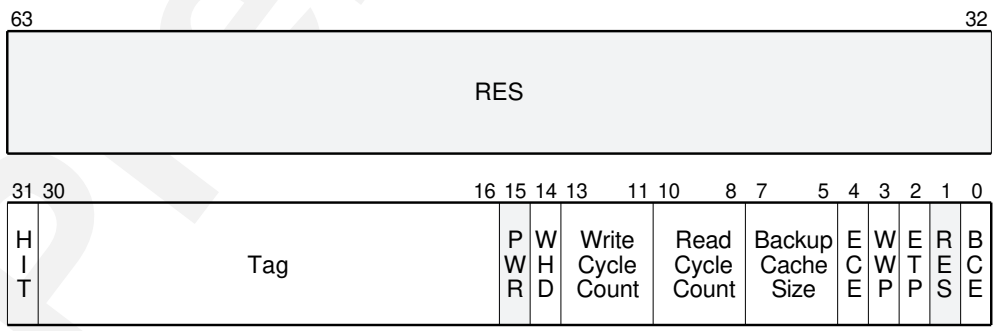
*DRAM page mode hit + DRAM page mode miss does not equal all DRAM cycles because page mode miss does not include DRAM accesses when **mem_ras_l** was already deasserted.

†Idle means not accessing Bcache or DRAM, or doing a DRAM refresh or VRAM shift register load.

5.5.7 Cache Register (CAR)

The Cache Register is used to configure the optional external Bcache. This register is read/write. All bits of this register are undefined at reset except for Bcache Enable, which is cleared.

Figure 5–12 Cache Register (CAR)



Preliminary

Table 5–13 Cache Register (CAR) Bit Description

Bits	Field	Type	Description
0	BCE	RW	Bcache Enable. This bit enables the Bcache. It is set at reset.
1	RES	RAX/ IGN	Reserved
2	ETP	RW	Enable Tag Parity Check. This bit enables parity checking for the Bcache.
3	WWP	RW	Write Wrong Tag Parity. This bit causes the Tag parity to be written incorrectly; used for diagnostics.
4	ECE	RW	Enable Bcache ECC. This bit enables error detection and correction on reads that hit in the Bcache.
7:5	Bcache Size	RW	This field indicates the size of the Bcache according to Table 5–14. The physical address is used as index and tag according to Figure 5–13. It is cleared at reset.
10:8	Read Cycle Count	RW	This field defines the read speed of the Bcache in cycles; measured from address driven from the chip to cache data latched on the chip. The number of cycles is the value of this field plus 3. This corresponds to timing diagram parameter T_{ca}. This field is cleared at reset.
13:11	Write Cycle Count	RW	This field defines the write speed of the Bcache in cycles; measured from data driven and **bc_we_l** asserted by the chip to **bc_we_l** deasserted by the chip. The number of cycles is the value of this field plus 1. This corresponds to timing diagram parameter T_{cw}.
14	WHD	RW	Write Hold Time. This bit determines the number of cycles of hold time cache write data is given after deassertion of **bc_we_l** (Section 5.6.17). 0 = 1 cycle; 1 = 2 cycles. This corresponds to timing diagram parameter T_{ch}.
15	PWR	RW	Power Saving. When set, this bit prevents the memory address lines from changing until a Bcache operation is detected. In addition, Bcache chip select is asserted only for Bcache operations. This adds one cycle of latency to the first read or write in a sequence of Bcache operations because the address lines are not speculatively updated. When this bit is cleared, the memory address lines are not held to a stable value in the absence of Bcache operations.

(continued on next page)

Preliminary

Table 5–13 (Cont.) Cache Register (CAR) Bit Description

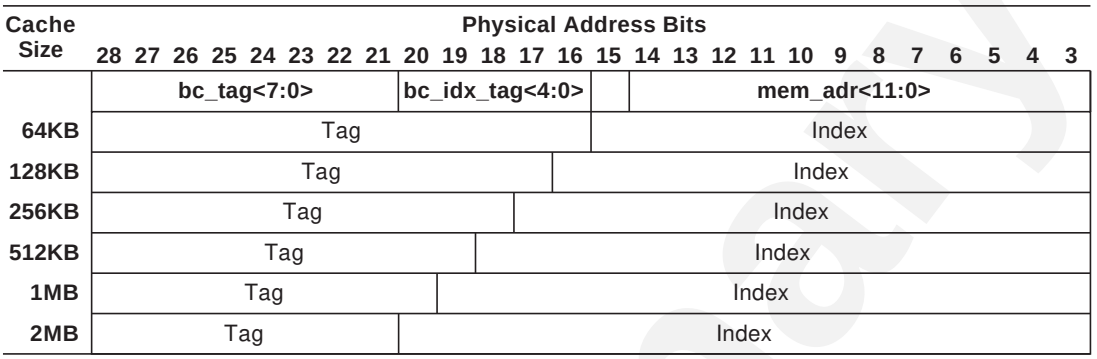
Bits	Field	Type	Description
30:16	Tag	RO	This field contains the tag value (PARITY, DIRTY, TAG<7:0>, IDX_TAG<4:0>) latched during the most recent data-stream Bcache lookup. It is intended for diagnostic use.
31	HIT	RO	Bcache Hit. This bit reflects the hit or miss status of the most recent data-stream Bcache Tag comparison. It is intended for diagnostic use.
63:32	RES	RAX/ IGN	Reserved

Table 5–14 External Cache Size Encodings

Cache Size	Code	Tag Address Bits	Index Address Bits
64KB	000	28:16	15:3
128KB	001	28:17	16:3
256KB	010	28:18	17:3
512KB	011	28:19	18:3
1MB	100	28:20	19:3
2MB	101	28:21	20:3
—	110	Reserved code, not used.	
—	111	Reserved code, not used.	

Preliminary

Figure 5–13 Cache Address Partition



Notes for Figure 5–13:

- Address bits <2:0> are not used since cache reference is quadword aligned.
- Address <14:3> is driven on **mem_addr<11:0>** pins (Section 5.6.3).
- Address bit 15 is driven on **bc_index** pin (Section 5.6.14)
- Address <20:16> are driven on **bc_idx_tag<4:0>** pins (Section 5.6.15) and are used as index or tag according to Table 5–20.
- Address <28:21> are driven on **bc_tag<7:0>** pins (Section 5.6.11).

5.5.8 Video and Graphics Control Register (VGR)

The Video and Graphics Control Register is used to configure the Video Memory and control graphics operations.

All bits of this register are undefined at reset and must be configured for proper operation.

Preliminary

Figure 5–14 Video and Graphics Control Register (VGR)

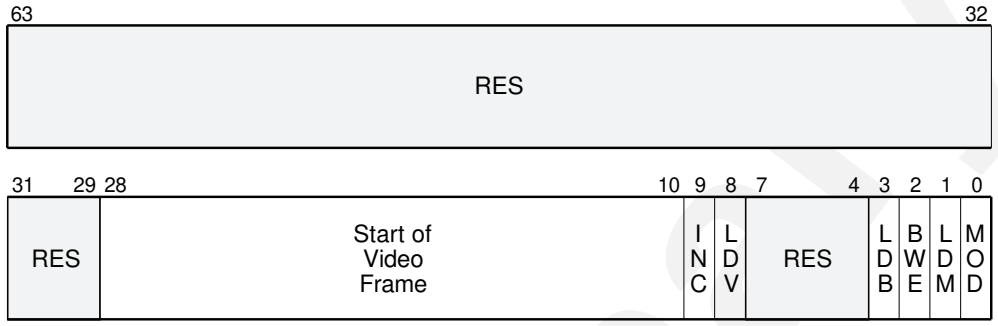


Table 5–15 Video and Graphics Control Register (VGR) Bit Description

Bits	Field	Type	Description
0	MOD	WO	Graphics Mode. This bit determines the type of memory operation caused by a write to Graphics Memory Space. 0 = Simple Mode; 1 = Transparant Stipple Mode.
1	LDM	WO	Load Mode. This bit controls the conditional loading of the Mode bit. A write to VGR with this bit value equal to 1 will also load the Mode bit; a write with this bit value equal to 0 will leave the Mode bit unaffected.
2	BWE	WO	Byte Write Enable. During graphics operations this bit will enable the Memory Controller to provide a byte mask on the **mem_ecc<7:0>** pins for writes to banks which have byte write enable set in the BCR.
3	LDB	WO	Load Byte Write Enable. This bit controls the conditional loading of the BWE bit. A write to VGR with this bit value equal to 1 will also load the Byte Write Enable bit; a write with this bit value equal to 0 will leave the Byte Write Enable bit unaffected.
7:4	RES	IGN	Reserved
8	LDV	WO	Load Video Control. This bit controls the conditional loading of the Frame Buffer Start, Address Increment, and VRAM Bank Select Fields. A write to VGR with this bit value equal to 1 will also load those fields; a write with this bit value equal to 0 will leave them unaffected.

(continued on next page)

Preliminary

Table 5–15 (Cont.) Video and Graphics Control Register (VGR) Bit Description

Bits	Field	Type	Description
9	INC	WO	Address Increment. After a shift register load operation, the Video Display Pointer (Section 5.5.8.1) must be incremented to the addresss of the next display information. This value selects which bit of the Video Display Pointer is to be incremented according to Table 5–16. The bit location that is incremented depends on the position of the VRAM's Column MSB location.
28:10	Start of Video Frame	WO	Frame Location. This field is bits 28:10 of the byte address of the base of the video frame buffer in memory (bits 9:0 of the address are implied to be 0—i.e.the frame buffer is aligned to 1 Kbyte). The frame base address is loaded into the Video Display Pointer (Section 5.5.8.1) when the external frame interrupt (**vframe_l** - Section 5.6.19) is asserted.
63:29	RES	IGN	Reserved

Table 5–16 VRAM Address Increment Encodings

VRAM Bank Size (quadwords)	Code	Address Bit Incremented
128K	0	10
256K or 512K	1	11

5.5.8.1 Video Display Pointer

The Video Display Pointer is a loadable counter which supplies the VRAM address during Video Shift Register Transfer Cycles.

When **vframe_l** (Section 5.6.19) is asserted the Video Display Pointer is loaded from the VGR Start of Video Frame Field, used as the address for a Full Video Shift Register Transfer Cycle and then incremented twice according to the VGR Increment field (the increment is done at either bit 10 or bit 11).

When **vrefresh_l** (Section 5.6.20) is asserted the Video Display Pointer is used as the address for a Split Video Shift Register Transfer Cycle and then incremented once according to the VGR Increment field.

The address is multiplexed according to the VRAM bank size as defined in the BCR Row Address Select Field (Section 5.5.1). Note that bits <9:0> of the Video Display Register are fixed at zero.

Preliminary

5.5.9 Plane Mask Register (PLM)

The Plane Mask register dictates which bits are going to be written during a write or read-modify-write operation to the graphics address space. Although this is a 64 bit register, software must write the same value to the high longword of the Plane Mask Register as it writes to the low longword, otherwise plane masked operations will be unpredictable.

All bits of this register are undefined at reset and must be configured for proper graphics operation.

Figure 5–15 Plane Mask Register (PLM)

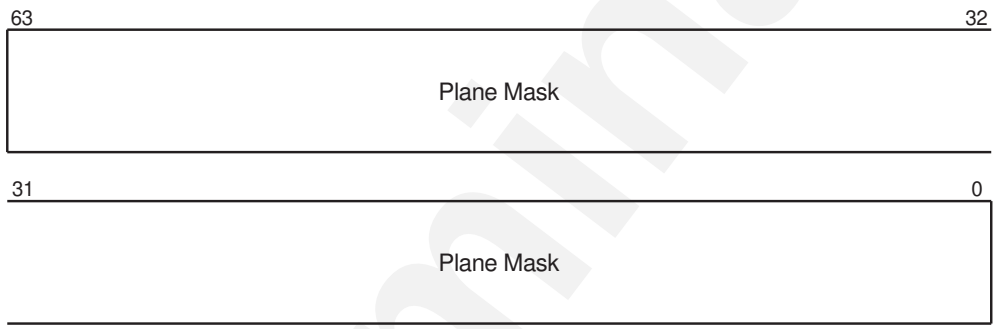


Table 5–17 Plane Mask Register (PLM) Bit Description

Bits	Field	Type	Description
63:0	Plane Mask	WO	The bits in this field select which bits are going to be written during a write or read-modify-write operation.

5.5.10 Foreground Register (FOR)

The Foreground register defines the foreground color for transparent stipple operations. Software must replicate the foreground value across the 32 bits of the foreground register (Foreground bit 0 controls bits 0 and 32, etc).

All bits of this register are undefined at reset and must be configured for proper video operation.

Preliminary

Figure 5–16 Foreground Register (FOR)

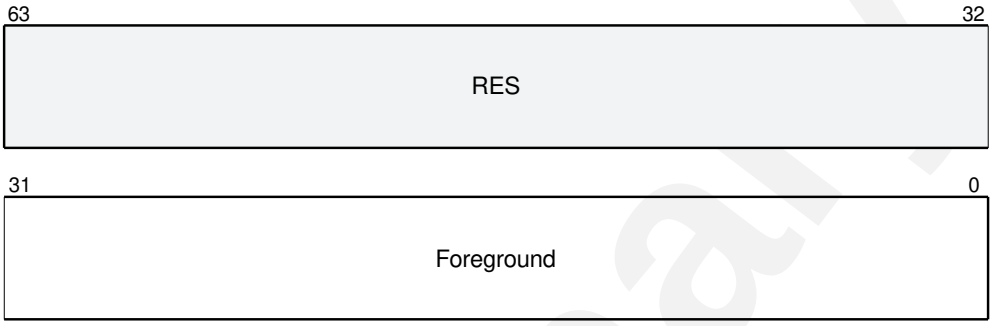


Table 5–18 Foreground Register (FOR) Bit Description

Bits	Field	Type	Description
31:0	Foreground	WO	Foreground value for Transparent Stipple Mode.
63:32	RES	IGN	Reserved

5.6 Memory Controller Pins

Functional descriptions of each pin are given below.

5.6.1 mem_data<63:0>

Signals **mem_data<63:0>** transmit data to and from the memories or Bcache and Memory Controller.

During a read from the Bcache, **mem_data<63:0>** are driven by the Bcache when **bc_oe_l** is asserted.

During a read from memory, **mem_data<63:0>** are driven by the memories (or tranceiver) when **mem_cas_l** and **mem_rd_oe** are asserted and **mem_write_l** is not asserted.

During a write **mem_data<63:0>** are driven with write data by the Memory Controller and are valid when either **bc_we_l** or **mem_cas_l**, **mem_wr_oe_l**, and **mem_write_l** are asserted.

During a Write-per-bit operation, **mem_data<63:0>** are driven with the Write per-bit Mask by the Memory Controller and are valid when **mem_ras_l**, **mem_wr_oe_l**, and **mem_write_l** are asserted.[1]

[1] The term **mem_ras_l** is equivalent to the **mem_rasa_l<3:0>** or **mem_rasb_l<3:0>** signal for the selected bank, as described in 5.5.1 and 5.5.2.

Preliminary

The **mem_data<63:0>** lines are weakly pulled down when no memory operation is in progress.

5.6.2 mem_ecc<7:0>

The **mem_ecc<7:0>** signals transmit the Error Correction Codes (ECC) to and from the Memories or Bcache and Memory Controller. ECC is generated by the Memory Controller for writes and checked by the Memory Controller for reads. The **mem_ecc<7:0>** signals have the same external timing as **mem_data<63:0>**. (Note that memory storage for ECC is optional, and the ECC checking may be disabled using the Bank Configuration Registers Section 5.5.1.) See Figure 5–2 for the correspondence of **mem_ecc<7:0>** to **mem_data<63:0>**.

The **mem_ecc<7:0>** signals also transmit a write byte mask for banks which have Byte Write enabled in the Bank Configuration Register (Section 5.5.1). Note that external logic is required to gate the DRAM write signals. When used in this way, **mem_ecc0** corresponds to **mem_data<7:0>**, **mem_ecc1** to **mem_data<15:8>**, etc. The mask is active high—a '1' means write the byte; a '0' inhibits writing.

The **mem_ecc<7:0>** lines are weakly pulled down when no memory operation is in progress.

5.6.3 mem_addr<11:0>

Signals **mem_addr<11:0>** transmit the Row and Column address to the memories and the lower 12 bits of the Bcache Index to the BCache. For memory reads and writes (ie not refresh) **mem_addr<11:0>** contains a valid Row address when **mem_ras_l** is asserted and a valid Column address when **mem_cas_l** is asserted.

5.6.4 mem_write_l

Signal **mem_write_l** provides the read/write control for the memories and enables loading of the Write-Per-Bit Mask. Signal **mem_write_l** is asserted prior to the assertion of **mem_cas_l** when the memory devices are to perform a write operation. Signal **mem_write_l** is deasserted prior to the assertion of **mem_cas_l** when the memory devices are to do a read operation.

The Write-Per-Bit function is used to write selected bits to the memories. The Write-Per-Bit function is activated in the DRAM/VRAMs when **mem_write_l** is asserted prior to **mem_ras_l** assertion. The value on **mem_data<63:0>** is loaded into the memory's internal write mask latch. Data bits that are low as **mem_ras_l** asserts will inhibit the subsequent write operations to that bit. Data bits that are high as **mem_ras_l** asserts will be modified during the write operation. When **mem_write_l** is high as **mem_ras_l** asserts, the

Preliminary

Write-Per-Bit capability is not selected and the write operation is performed on all data bits.

5.6.5 mem_rasa_l<3:0> and mem_rasb_l<3:0>

Each pair of **mem_rasa_l<3:0>** and **mem_rasb_l<3:0>** are associated with a bank of memory. During a normal read or write, the assertion of **mem_ras_l**[1] indicates that **mem_addr<11:0>** contains a valid Row Address. Which RAS is asserted depends on which bank is addressed whether or not split bank is enabled in the BCR (Section 5.5.1) and the row address bit shown in Table 5–5.

During a memory refresh cycle, all **mem_rasa_l<3:0>** and **mem_rasb_l<3:0>** outputs are asserted together (refresh cycles are the CAS-before-RAS type) provided that the Refresh Enable bit in the GTR is set (Section 5.5.4).

During VRAM shift register loads, both **mem_rasa_l*n*** **mem_rasb_l*n*** for bank *n* will be asserted.

5.6.6 mem_cas_l

When asserted during memory reads and writes, **mem_cas_l** indicates that **mem_addr<11:0>**, **mem_data<63:0>** (for writes), and **mem_write_l** contain valid information.

During a memory refresh cycle, **mem_cas_l** is asserted before **mem_ras_l** is asserted.

5.6.7 mem_dtoe_l

This multi-function pin controls the output enable function for memories or the Shift Register Load function for VRAMs.

For normal memory operations this pin will be deasserted when **mem_ras_l** is asserted. During a read cycle, this pin is asserted prior to **mem_cas_l** assertion to enable the memory's data output drivers to turn on.

To indicate a Shift Register Load sequence to VRAMs, this signal is asserted prior to **mem_ras_l** assertion. Either a Full or Split Register Load sequence will be done depending on the value of **mem_dsf**.

Preliminary

5.6.8 mem_dsf

The Special Function Select is used to determine if a Full or Split Shift Register Load is to be done as indicated in Table 5–19. This signal is valid prior to **mem_ras_l** assertion when **mem_dtoe_l** is asserted.

Table 5–19 DSF Function

DTOE_L	DSF	Function at RAS Assertion
1	X	Normal Read or Write
0	0	Full Shift Register Load
0	1	Split Shift Register Load

A Full Shift Register Load cycle will load an entire row of memory data into the VRAM's Shift Register. A Split Shift Register Load cycle will load only one-half of a row into one-half of the VRAM's Shift Register. The most significant bit of the column address determines which half of the row is to be loaded (some VRAMs will control this internally). The remaining column address bits will be zeros.

5.6.9 mem_rd_oe

Signal **mem_rd_oe** enables the optional external memory transceiver to drive data from the memory parts onto **mem_data<63:0>** and **mem_ecc<7:0>**. It is asserted during read cycles when **mem_cas_l** is asserted. If no transceiver is used, this pin should be ignored.

5.6.10 mem_wr_oe_l

Signal **mem_wr_oe_l** enables the optional external memory transceiver to drive data from **mem_data<63:0>** and **mem_ecc<7:0>** to the memory parts. It is asserted during write cycles when **mem_data<63:0>** is driven. If no transceiver is used, this pin should be ignored.

5.6.11 bc_tag<7:0>

Signals **bc_tag<7:0>** represent the upper part of the physical address stored in a cache block. During a Bcache lookup, the cache will drive these bits (and possibly **bc_idx_tag<4:0>**; see Section 5.6.15) with the tag corresponding to the current index being driven on **mem_addr<11:0>**, **bc_index**, and the nontag bits of **bc_idx_tag<4:0>**. During a BCache fill or write, the Memory Controller will drive the upper bits of the address onto these lines to be stored in the Bcache tag array.

The **bc_tag<7:0>** lines are weakly pulled down when no memory operation is in progress.

Preliminary

5.6.12 bc_dirty

Signal **bc_dirty** represents the status of the data stored in a cache block. A value of '0' indicates that the cache and memory contain the same data; a value of '1' indicates that the cache contains more recently written data than memory (ie memory data is stale). It has the same timing characteristics as **bc_tag<7:0>**.

The **bc_dirty** line is weakly pulled down when no memory operation is in progress.

5.6.13 bc_parity

Signal **bc_parity** transmits the Bcache tag parity to the tag array during cache fills and writes and from the tag array during cache lookups. It has the same timing characteristics as **bc_tag<7:0>**.

The **bc_parity** line is weakly pulled down when no memory operation is in progress.

5.6.14 bc_index

Signal **bc_index** transmits Bcache index bit 12 to the cache.

5.6.15 bc_idx_tag<4:0>

Each bit of **bc_idx_tag<4:0>** can act as either an index bit or a tag bit, depending on the size of the Bcache. The function of each bit is determined by the size of the Bcache as shown in Table 5–20.

Table 5–20 Use of bc_idx_tag

Bcache Size	bc_idx_tag<4:0>				
	4	3	2	1	0
2MB	Index 17	Index 16	Index 15	Index 14	Index 13
1MB	Tag 8	Index 16	Index 15	Index 14	Index 13
512KB	Tag 8	Tag 9	Index 15	Index 14	Index 13
256KB	Tag 8	Tag 9	Tag 10	Index 14	Index 13
128KB	Tag 8	Tag 9	Tag 10	Tag 11	Index 13
64KB	Tag 8	Tag 9	Tag 10	Tag 11	Tag 12

If the function of the bit is index, then its direction is output only, otherwise its direction depends on the cycle type; input during a cache lookup; output during a cache fill or write.

Preliminary

The **bc_idx_tag<4:0>** lines are weakly pulled down when no memory operation is in progress.

5.6.16 bc_cs_l

Signal **bc_cs_l** is the chip select for the Bcache SRAMs. They are enabled when this signal is asserted and disabled otherwise. This signal provides a power saving feature when no cache accesses are being done.

5.6.17 bc_we_l

Signal **bc_we_l** is asserted to write the data on **mem_data<63:0>** and **mem_ecc<7:0>** into the Bcache Data SRAM array, and the tag on **bc_tag<7:0>**, **bc_idx_tag<4:0>**, **bc_dirty** and **bc_parity** to be written into the tag SRAMs.

5.6.18 bc_oe_l

Signal **bc_oe_l** is the output enable for the Bcache. It enables the Bcache SRAM array to drive data on **mem_data<63:0>** and **mem_ecc<7:0>**, and Tags onto **bc_tag<7:0>**, **bc_idx_tag<4:0>**, **bc_dirty** and **bc_parity**.

5.6.19 vframe_l

The assertion of **vframe_l** causes the Memory Controller to:

1. Reload the Video Display Pointer (Section 5.5.8.1) with the value stored in the Start of Video Frame field of the VGR (Section 5.5.8).
2. Perform a Full Shift Register Load cycle to the VRAM bank (selected by the Start of Video Frame Field of the VGR (Section 5.5.8)).
3. Increment the Video Display Pointer twice as specified by the Address Increment field of the VGR.

Signal **vframe_l** is edge sensitive. The assertion (falling) edge is captured by an edge detection circuit; the output of which is synchronized and allowed to settle before being used. The edge detection circuit is rearmed when the Shift Register Load cycle is done. A subsequent assertion of **vframe_l** or of **vrefresh_l** prior to the completion of the Shift Register Load cycle will be lost.

5.6.20 vrefresh_l

The assertion of **vrefresh_l** causes the Memory Controller to:

1. Perform a Split Shift Register Load cycle to the VRAM bank (selected by the Start of Video Frame Field of the VGR (Section 5.5.8)).
2. Increment the Video Display Pointer (Section 5.5.8.1) as specified by the Address Increment field of the VGR.

Preliminary

Signal **vrefresh_l** is edge sensitive. The assertion (falling) edge is captured by an edge detection circuit; the output of which is synchronized and allowed to settle before being used. The edge detection circuit is rearmed when the Shift Register Load cycle is done. A subsequent assertion of **vrefresh_l** or of **vframe_l** prior to the completion of the Shift Register Load cycle will be lost.

5.7 Memory Cycles

In the following timing diagrams the parameters decribed in Table 5–21 are referred to. These times are programmed into the Bank Timing Registers (Section 5.5.3), Global Timing Register (Section 5.5.4), and Cache Register (Section 5.5.7). The timing diagrams assume all parameters are set to minimum values—larger values can be accounted for by inserting the programmed number of cycles with the signals held constant at the levels they have reached at the end of the interval (ie a longer time on T_{01} defers when T_1 happens—the signals maintain the value they changed to at T_0).

Table 5–21 Memory Timing Parameters

Parameter	Events
T_{ca}	Bcache Access Time
T_{cw}	Bcache Write Time
T_0	Drive Row Address on **mem_addr<11:0>**. Drive Write Mask on **mem_data<63:0>**.
T_1	Assert **mem_ras_l** for selected bank, tristate **mem_data<63:0>** for read.
T_2	Drive Column Address on **mem_addr<11:0>**; if write cycle drive **mem_data<63:0>**. See Note 1.
T_3	Assert **mem_cas_l**.
T_4	Deassert **mem_cas_l**; if read cycle latch **mem_data<63:0>**; if page mode cycle drive next Column Address on **mem_addr<11:0>**; if page mode write cycle drive **mem_data<63:0>**; if not page mode cycle deassert **mem_ras_l**.
T_5	Assert **mem_cas_l**.
T_6	Deassert **mem_cas_l**; if read cycle latch **mem_data<63:0>**; tristate **mem_data<63:0>**; deassert **mem_ras_l**. See Note 2.
T_{minRAS}	**mem_ras_l** assertion time during refresh cycle.
$T_{RASprech}$	RAS and CAS precharge time.

Preliminary

Notes for Table 5–21

1. When going from page mode read to page mode write, the Memory Controller will wait t_{OFF} (Section 5.5.3) before driving **mem_data<63:0>** and **mem_ecc<7:0>**. If t_{OFF} is equal to or longer than T_{45} (CAS precharge) then CAS precharge will be T_{45} + 1 cycles long and **mem_data<63:0>** and **mem_ecc<7:0>** will be driven one cycle before **mem_cas_l** assertion.)
2. **mem_ras_l** will deassert at T_6 if there is another memory cycle pending which is not a page mode cycle.)

The memory cycle performed is a function of the type of memory operation required and the state of the Memory Controller at the time the operation is required. Table 5–22 describes the operations and what conditions cause them to happen.

Table 5–22 Memory Operations

Type of Operation	Originated By
DRAM Refresh	Internal Refresh Interval Counter Overflow (see Section 5.5.4).
Full Video Shift Register Transfer Cycle	Assertion of **vframe_l** pin (see Section 5.6.19).
Split Video Shift Register Transfer Cycle	Assertion of **vrefresh_l** pin (see Section 5.6.20).
Read (Cache or DRAM)	Any of the following: • Instruction Fetch. • Load Instruction. • Store Longword instruction to a bank that does not support write-per-bit or byte write operations (Read-modify-write). • PCI Device DMA Read. • PCI Device DMA Write of less than a full Quadword to a bank that does not support write-per-bit or byte write operations (Read-modify-write).

(continued on next page)

Preliminary

Table 5–22 (Cont.) Memory Operations

Type of Operation	Originated By
Write (Cache or DRAM)	Any of the following: • Store • PCI Device DMA Write
Victim Write (DRAM)	All of the following: • CPU Read or Write to Cacheable Space, and • Cache Miss (Tag does not match), and • Block is Dirty.
RAS Precharge	All of the following (one or more event may occur simultaneously): • Read or Write on different DRAM page than the one for which **mem_ras_l** was previously asserted will happen next. • Write-per-bit with a different mask than the one for which **mem_ras_l** was previously asserted will happen next. • Refresh cycle will happen next. • Video Shift Register load (either full or split) will happen next. • MAX RAS Assertion period has expired.

An idle cycle occurs when none of the other cycle types in Table 5–22 is required. No Memory Controller signals change during the idle cycle with the exception of **mem_addr<11:0>**, **bc_index**, and **bc_idx_tag<4:0>** which are acting as index. (This is done because the index is being driven out in anticipation of an internal cache miss to make the Bcache latency as small as possible.) **mem_ras_l** may be either an asserted or deasserted state, depending on whether or not a page mode cycle has occured recently.[1] All other memory control signals are deasserted. The idle cycle is one cycle long, but multiple idle cycles may occur in succession.

Preliminary

Table 5–23 defines the conditions which determine when reads or writes will access the Bcache, use page-mode, write-per-bit, and byte write.

Table 5–23 Memory Read/Write Options

Option	When Used
Cacheable	Bcache enabled (Section 5.5.1) and Address is in Cacheable Space (Table 2–5).
Page Mode Access	Address is on same DRAM page as previous cycle (Section 5.2.2) and Max RAS Assertion time has not expired since **mem_ras_l** assertion (Section 5.5.4).
Write-per-bit	Addressed memory bank has write-per-bit enabled (Section 5.5.1) and Address is in Graphics Space (Table 2–5).
Byte Write	Address is in Graphics Space (Table 2–5), addressed memory bank has byte write enabled (Section 5.5.1), and Byte write bit is set in VGR (Section 5.5.8).

Preliminary

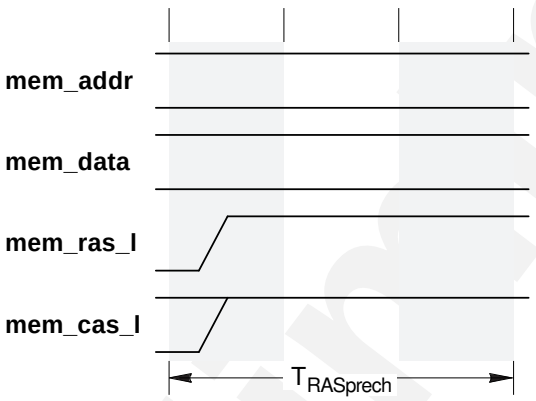
5.7.1 RAS Precharge Timing

RAS Precharge cycle is done when **mem_ras_l** has been asserted and one or more of the conditions in Table 5–22 is met.

The sequence is:

1. **mem_ras_l** is deasserted; **mem_cas_l**, if it had been asserted, is deasserted.
2. Wait for T $_{RASprech}$ cycles.

Figure 5–17 RAS Precharge Timing



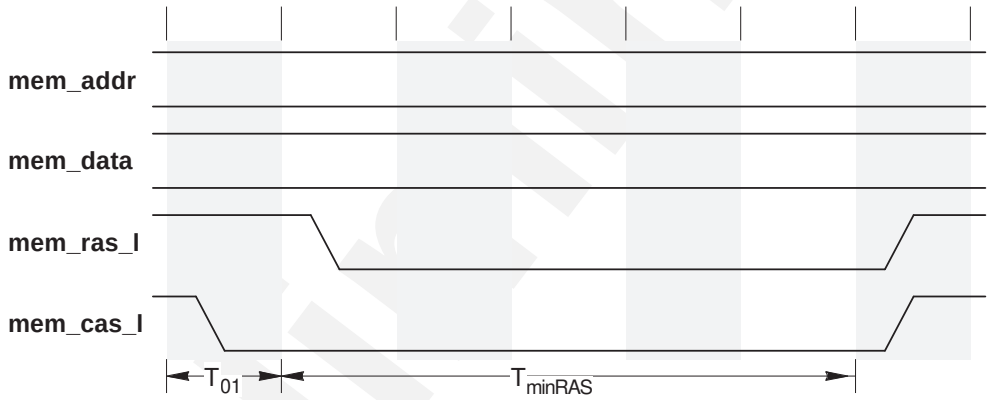
Preliminary

5.7.2 DRAM Refresh Timing

DRAM Refresh cycle is done when the Refresh Interval in the Global Timing Register (Section 5.5.4) expires. The Memory Controller coordinates refresh cycles with reads and writes, such that burst activities such as internal cache fills are not disrupted by refresh cycles. This helps to maximize memory bandwidth and minimize latency. The sequence is:

1. **mem_cas_l** is asserted.
2. After T_{01} cycles all **mem_ras_l** are asserted.
3. After T $_{minRAS}$ cycles **mem_ras_l** and **mem_cas_l** are deasserted.

Figure 5–18 DRAM Refresh Timing



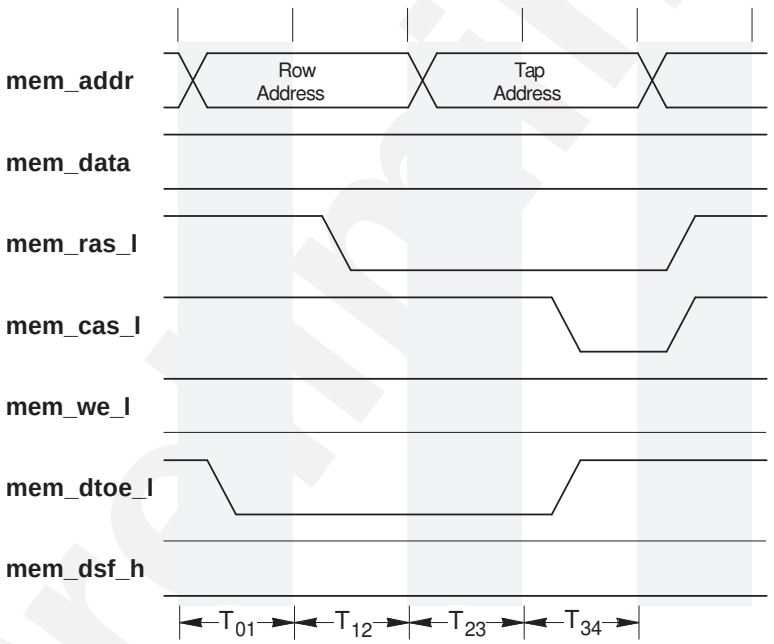
Preliminary

5.7.3 Full Video Shift Register Transfer Timing

Full Video Shift Register Transfer cycle is done in response to assertion of the **vframe_l** input (Section 5.6.19). The sequence is:

1. Row address is driven on **mem_addr<11:0>**, **mem_dtoe_l** is asserted.
2. After T_{01} cycles **mem_ras_l** is asserted.
3. After T_{12} cycles tap address is driven on **mem_addr<11:0>**.
4. After T_{23} cycles **mem_dtoe_l** is deasserted and **mem_cas_l** is asserted.
5. After T_{34} cycles **mem_ras_l**, **mem_cas_l**, and **mem_dtoe_l** are deasserted.

Figure 5–19 Full Video Shift Register Transfer Timing



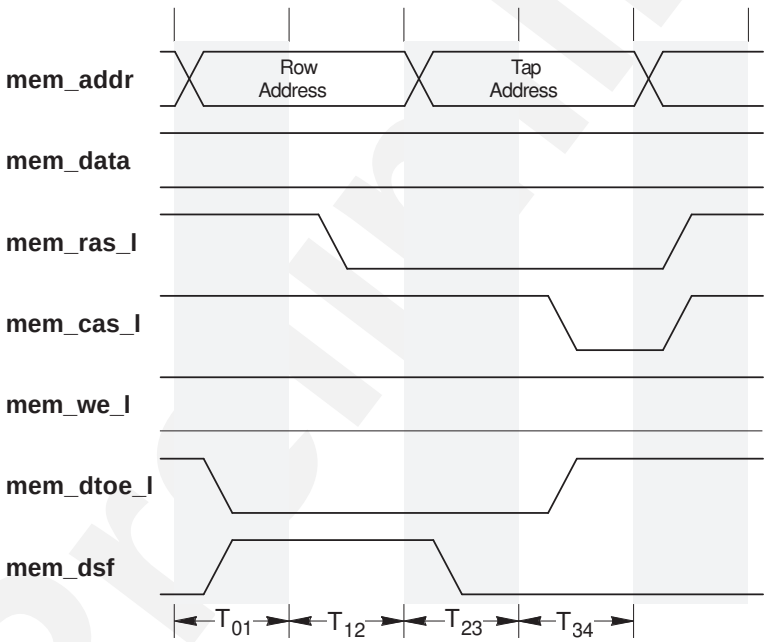
Preliminary

5.7.4 Split Video Shift Register Transfer Timing

Split Video Shift Register Transfer cycle is done in response to assertion of the **vrefresh_l** input (Section 5.6.20). The sequence is:

1. Row address is driven on **mem_addr<11:0>**, **mem_dtoe_l** is asserted, **mem_dsf** is asserted.
2. After T_{01} cycles **mem_ras_l** is asserted.
3. After T_{12} cycles tap address is driven on **mem_addr<11:0>**, **mem_dsf** is deasserted.
4. After T_{23} cycles **mem_dtoe_l** is deasserted and **mem_cas_l** is asserted.
5. After T_{34} cycles **mem_ras_l**, **mem_cas_l**, and **mem_dtoe_l** are deasserted.

Figure 5–20 Split Video Shift Register Transfer Timing



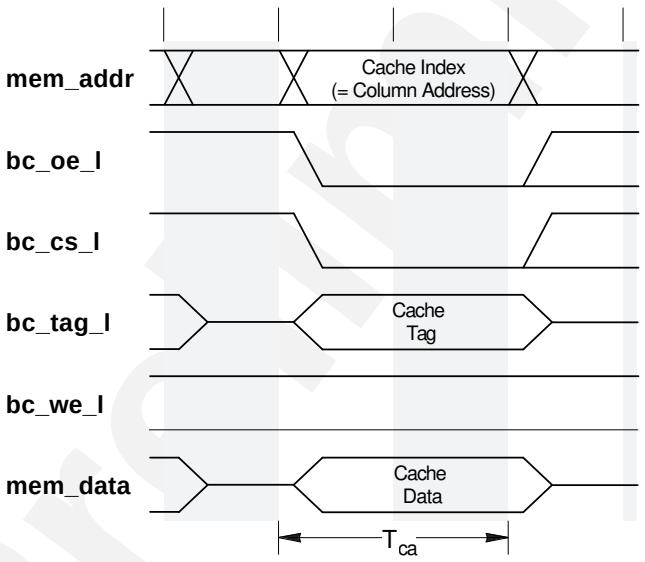
Preliminary

5.7.5 Bcache Read Timing

Bcache Read Cycle may occur when there is a read required from one of the operations in Table 5–22. The Bcache will be read first if the conditions for Cacheable cycle in Table 5–23 are met. The sequence is:

1. **mem_data<63:0>**, **mem_ecc<7:0>**), **bc_tag_l**, **bc_dirty** and **bc_parity** are tristated; cache index is driven on **mem_addr<11:0>**; **bc_oe_l** and **bc_cs_l** are asserted.
2. After T_{ca} cycles **mem_data<63:0>** and **mem_ecc<7:0>** are latched from the cache data RAMs; **bc_tag_l**, **bc_dirty** and **bc_parity** are latched from the cache tag RAMs; **bc_oe_l** and **bc_cs_l** are deasserted.
3. The tag and parity are checked internally. If the tag matches the physical address the read is considered a hit.

Figure 5–21 Bcache Read Timing



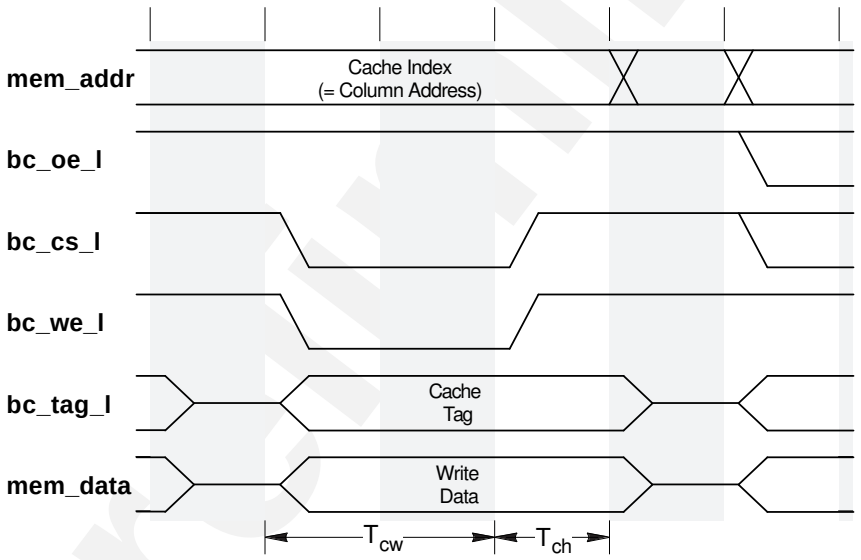
Preliminary

5.7.6 Bcache Write Timing

Bcache Write Cycle may occur when there is a write required from one of the operations in Table 5–22, or a read which will allocate a block. On a cache write the index is set up in advance from either a Bcache Read or a DRAM Read. The sequence is:

1. **mem_data<63:0>**, **mem_ecc<7:0>**, **bc_tag_l**, **bc_dirty** and **bc_parity** are driven; **bc_oe_l** is deasserted, **bc_we_l** and **bc_cs_l** are asserted. **bc_dirty** is asserted for a write; deasserted for a read.
2. After T $_{cw}$ cycles **bc_we_l** and **bc_cs_l** are deasserted.
3. After one or two cycles, depending on the value in the Cache Register hold time bit **mem_addr<11:0>**, **bc_index**, **bc_idx_tag<4:0>** are changed.

Figure 5–22 Bcache Write Timing



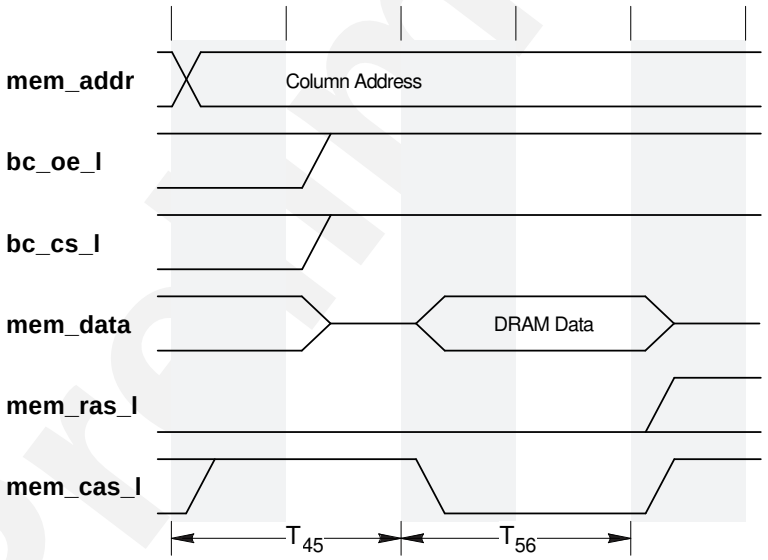
Preliminary

5.7.7 Page Mode Read Timing

Page Mode Read Cycle will occur when there is a read required from one of the operations in Table 5–22 and the conditions for Page Mode in Table 5–23 are met. The sequence is:

1. Column Address is driven on **mem_addr<11:0>**, **mem_dtoe_l** is asserted. **mem_data<63:0>** and **mem_ecc<7:0>** are tristated. If Bcache was just previously accessed **bc_oe_l** and **bc_cs_l** are deasserted. Note that in the case when the Bcache was accessed, the cache index driven onto **mem_addr<11:0>** is the same as the column address, and the first cycle of Figure 5–23 corresponds to the last cycle of the Bcache read (Figure 5–21).
2. After T_{45} cycles **mem_cas_l** is asserted.
3. After T_{56} cycles **mem_data<63:0>** and **mem_ecc<7:0>** are latched from the DRAMs and **mem_cas_l** is deasserted. If the conditions for RAS precharge in Table 5–22 are met then **mem_ras_l** is deasserted.

Figure 5–23 Page Mode Read Timing



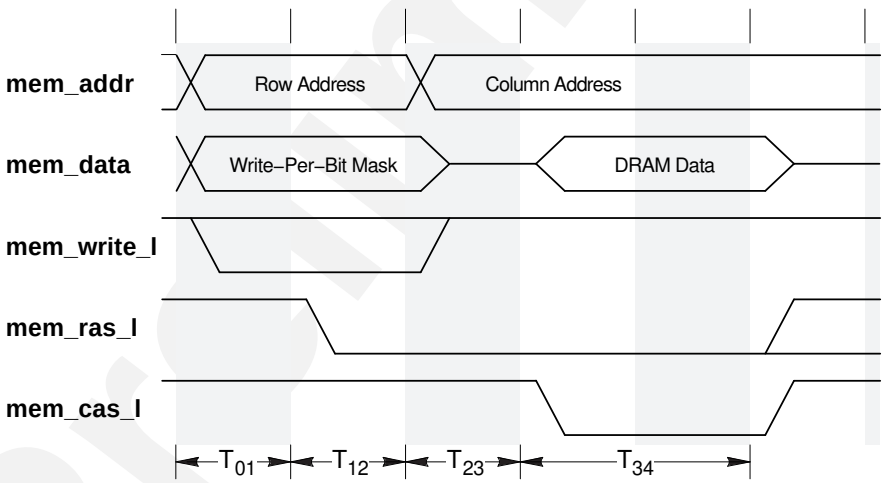
Preliminary

5.7.8 Nonpage Mode Read Timing

Nonpage Mode Read Cycle may occur when there is a read required from one of the operations in Table 5–22 and the conditions for Page Mode in Table 5–23 are not met. The sequence is:

1. Row Address is driven on **mem_addr<11:0>**. If the conditions in Table 5–23 for write-per-bit are met then Write Mask is driven on **mem_data<63:0>** and **mem_write_l** is asserted.
2. After T_{01} **mem_ras_l** is asserted.
3. After T_{12} Column Address is driven on **mem_addr<11:0>**. **mem_data<63:0>** and **mem_ecc<7:0>** are tristated.
4. After T_{23} cycles **mem_cas_l** is asserted.
5. After T_{34} cycles **mem_data<63:0>** and **mem_ecc<7:0>** are latched from the DRAMs and **mem_cas_l** is deasserted. If the conditions for RAS precharge in Table 5–22 are met then **mem_ras_l** is deasserted.

Figure 5–24 Nonpage Mode Read Timing



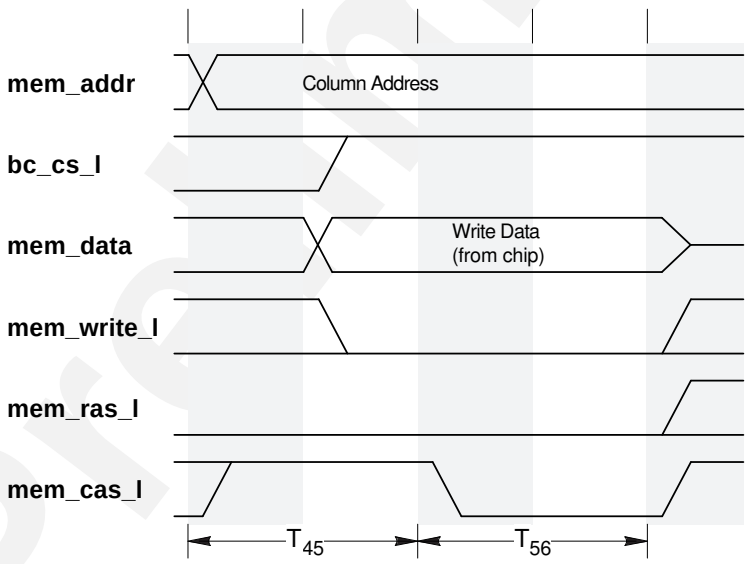
Primary

5.7.9 Page Mode Write Timing

Page Mode Write Cycle may occur when there is a write required from one of the operations in Table 5–22 and the conditions in Table 5–23 for Page Mode are met. The sequence is:

1. Column Address is driven on **mem_addr<11:0>**. Write data and ECC are driven on **mem_data<63:0>** and **mem_ecc<7:0>**. **mem_write_l** is asserted. If Bcache was just previously accessed **bc_oe_l** and **bc_cs_l** are deasserted. Note that in the case when the Bcache was accessed, the cache index driven onto **mem_addr<11:0>** is the same as the column address, and the first cycle of Figure 5–25 corresponds to the last cycle of the Bcache read (Figure 5–21).
2. After T_{45} cycles **mem_cas_l** is asserted.
3. After T_{56} cycles **mem_cas_l** and **mem_write_l** are deasserted. If the conditions for RAS precharge in Table 5–22 are met then **mem_ras_l** is deasserted.

Figure 5–25 Page Mode Write Timing



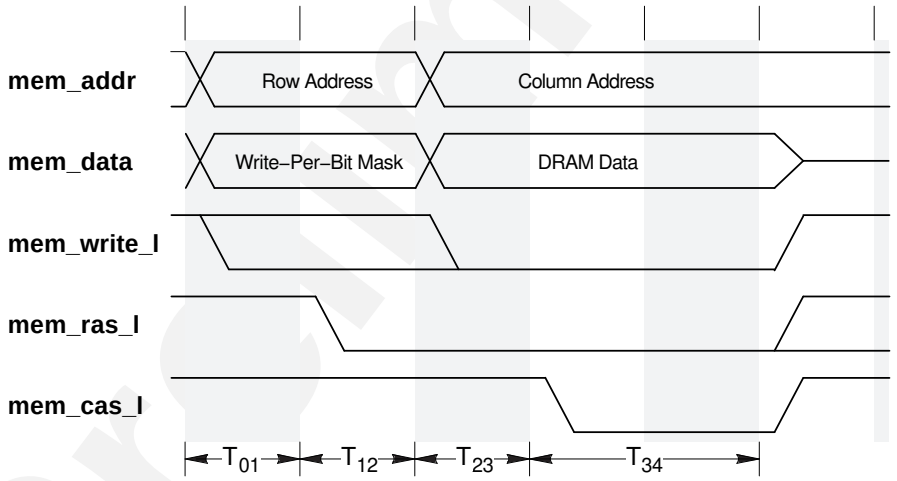
Preliminary

5.7.10 Nonpage Mode Write Timing

Nonpage Mode Write Cycle may occur when there is a write required from one of the operations in Table 5–22 and the conditions for Page Mode in Table 5–23 are not met. The sequence is:

1. Row Address is driven on **mem_addr<11:0>**. If the conditions in Table 5–23 for write-per-bit are met then Write Mask is driven on **mem_data<63:0>** and **mem_write_l** is asserted.
2. After T_{01} **mem_ras_l** is asserted.
3. After T_{12} Column Address is driven on **mem_addr<11:0>**. **mem_write_l** is asserted, and **mem_data<63:0>** and **mem_ecc<7:0>** are tristated.
4. After T_{23} cycles **mem_cas_l** is asserted.
5. After T_{34} cycles **mem_cas_l** is deasserted. If the conditions for RAS precharge in Table 5–22 are met then **mem_ras_l** is deasserted.

Figure 5–26 Nonpage Mode Write Timing



Preliminary

6
I/O Controller

This chapter describes the I/O controller (IOC) operation and its registers. This chapter does not replicate information that can be found in the *PCI Local Bus Specification, Revision 2.0*.

6.1 Introduction

The IOC is the interface between peripheral devices and the CPU and system memory. All peripheral devices in a system based on the DECchip 21066 or DECchip 21068 connect to the CPU and system memory through the IOC. The IOC interface protocol complies with the *PCI Local Bus Specification, Revision 2.0*. Peripheral chips also comply with the *PCI Local Bus Specification, Revision 2.0* can be connected to the IOC without any glue logic. However, external logic is required for interrupt arbitration, interrupt vector generation, and DMA request arbitration.

The DECchip 21066 and DECchip 21068 are *not* PCI peripherals. The IOC implements the functions of a bridge between the PCI, the CPU, and system memory. These functions are not sufficient to interface the chip as a PCI peripheral component.

6.1.1 Scatter Gather Mapping Support

The IOC incorporates scatter-gather mapping logic to translate 32-bit addresses generated by PCI bus masters to the 34-bit CPU physical address space. The IOC implements an 8-entry translation look-aside buffer (TLB) for fast translations. The scatter gather map tables are stored in system memory for low cost and are automatically read by the IOC when a translation does not hit in the TLB.

Preliminary

6.1.2 Sparse and Dense Memory Support

The IOC supports both sparse and dense memory space.

Sparse memory space is a region in which the lower bits of the CPU address indicate transfer size, and the address is shifted down by five bits during the translation from CPU address to PCI address. The host address extension register (IOC_HAE, Section 6.5.1) extends the shifted address to 32 bits. The size of transfers in this space can be byte, word, tribyte, longword, and quadword. The CPU references sparse address space when CPU address bits <33:32> = 10.

Dense memory space is a region in which only lonword and quadword transfers are supported. The CPU address is transparently mapped to the PCI address. This uses the CPU write buffers more efficiently but restricts transfer size capability. The CPU references sparse address space when CPU address bits <33:32> = 11. The IOC_HAE register is not required for references to this space because the address is not shifted.

6.1.3 PCI Parity Support

The IOC supports PCI bus parity generation and checking. The agent driving the PCI **ad<31:0>** lines on any given bus cycle is responsible for driving even parity on the PCI **par** signal pin. During all phases where address or data information is transferred on PCI parity covers the **ad<31:0>** and the **c_be_l<3:0>** signal pins. Bytes that are masked during a transfer are required to be driven to stable levels and are included in the parity computation. During configuration, special cycle or interrupt acknowledge cycles all address lines are required to be driven to stable levels and are included in the parity computation. The computed parity is driven onto the PCI **par** signal in the cycle following the cycle where the address or data information was transferred. PCI address and data parity errors are reported through different pins.

Note

The PCI **serr_l** signal is not connected directly to the chip.

The PCI signal **serr_l** is used to report all address parity errors and data parity errors during PCI special cycles. Parity errors detected by the IOC that would normally be reported on **serr_l** by PCI devices will be reported to the CPU via an internal interrupt. System designers can report the assertion of the PCI **serr_l** signal by PCI peripherals via the chip's interrupt signal pins. **serr_l** is an asynchronous PCI signal. **serr_l** will normally be asserted one cycle after valid parity is driven on **par** (two cycles after valid address or

Preliminary

data). PCI **serr_l** can also be used by peripherals to report other severe error conditions.

The PCI signal **perr_l** is used to report data parity errors on all transactions except during PCI special cycles. Only data parity errors are reported via **perr_l**. The IOC provides a signal pin for direct connection to the PCI **perr_l** signal. **perr_l** is synchronous to the PCI clock. Data parity errors are reported on **perr_l** one cycle after valid parity is driven on **par** (two cycles after valid data).

The IOC parity disable register (IOC_PAR_DIS, Section 6.5.5) allows parity checking and reporting to be disabled. When this register is written, the IOC will neither log nor report parity errors.

6.1.4 Exclusive Access Support

The IOC supports the PCI exclusive access protocol for peripheral initiated transfers with system memory (DMA). The chip cannot generate exclusive access cycles for CPU initiated PCI transfers. The granularity of the lock is all of physical memory.

6.1.5 Clocking

With the exception of the PCI reset signal, **rst_l**, all PCI signals are synchronous to the PCI clock signal and are sampled on the rising edge of the PCI clock. Timing parameters for PCI signals are specified relative to the rising edge of the PCI clock.

The IOC provides a PCI clock input. The PCI clock input pin is used to control the latching of PCI signals in the IOC and also to generate the control strobes for the IOCs outputs. The PCI clock input is asynchronous to the internal CPU clock. Synchronization between the PCI and CPU clocking domains occurs within the IOC.

6.2 CPU Initiated PCI Cycles

The IOC operates as a PCI master when the CPU executes a Load or Store instruction that addresses a PCI peripheral. The IOC provides address windows to the CPU that allow access to the memory, I/O, and configuration address spaces of PCI. The IOC provides a register that when read will generate a PCI interrupt acknowledge cycle, or, when written will generate a PCI special cycle. The address ranges for the various PCI address spaces supported by the IOC are listed in Table 6–1. The PCI command codes generated during each cycle type is also listed in Table 6–1.

Preliminary

Figure 6–1 shows the complete address translation for different cycle types.

Table 6–1 CPU Address Windows to PCI Address Spaces

CPU Address <33:0>	PCI Address Space	PCI Command Code c_be_l<3:0> Reads	Writes
1 Axxx xxxx 1 Bxxx xxxx	Interrupt Acknowledge	0000	N/A
1 Axxx xxxx 1 Bxxx xxxx	Special Cycle	N/A	0001
1 Cxxx xxxx 1 Dxxx xxxx	I/O Space	0010	0011
1 Exxx xxxx 1 Fxxx xxxx	Configuration	1010	1011
2 xxxx xxxx	Sparse Memory	0110	0111
3 xxxx xxxx	Dense Memory	0110	0111

Figure 6–1 CPU-to-PCI Address Translation

Read Write	CPU a<33:32>	CPU a<31:29>	PCI ad<31:27>	PCI ad<26:24>	PCI ad<23:2>	PCI ad<1:0>	PCI c_be_l<3:0>	Transaction
Write	10	Nonzero	HAE<31:27>*	A<31:29>	A<28:7>	00	0111	Sparse memory write
Write	10	000	00000	A<31:29>	A<28:7>	00	0111	Sparse memory write
Read	10	Nonzero	HAE<31:27>	A<31:29>	A<28:7>	00	0110	Sparse memory read
Read	10	000	00000	A<31:29>	A<28:7>	00	0110	Sparse memory read
Write	11	XXX	A<31:27>	A<26:24>	A<23:2>	00	0111	Dense memory write
Read	11	XXX	A<31:27>	A<26:24>	A<23:2>	00	0110	Dense memory read
Write	01	110	00000	000	A<28:7>	A<6:5>	0011	I/O write
Read	01	110	00000	000	A<28:7>	A<6:5>	0010	I/O read
Write	01	111	00000	000	A<28:7>	CFG<1:0>**	1011	Configuration write
Read	01	111	00000	000	A<28:7>	CFG<1:0>	1010	Configuration read
Write	01	101	HAE<31:27>	A<31:29>	A<28:7>	00	0001	Special cycle
Read	01	101	HAE<31:27>	A<31:29>	A<28:7>	00	0000	Interrupt acknowledge

*HAE = Host address extension register
**CFG = Configuration cycle type register

6.2.1 CPU Request Queue

The IOC implements a request queue for CPU initiated PCI transactions. The queue is used for all CPU initiated PCI transfers and for CPU access to registers internal to the IOC. Up to two CPU initiated PCI write transfers may be held in the request queue. Only one CPU initiated PCI read transfer or one

Preliminary

CPU access to an IOC register (read or write) may be held in the queue at any moment. A CPU initiated PCI read or a CPU access of an IOC register may be held in the second entry of the CPU request queue if the preceding entry in the queue is a CPU initiated PCI write transfer. Each entry in the queue provides storage for an address and up to quadword of write data.

6.2.2 Requesting PCI Mastership

When the IOC has a CPU initiated PCI read or write cycle queued, and its grant signal, **gnt_l** is not asserted, it will assert its bus request signal, **req_l**. Logic external to the CPU will arbitrate requests from various devices in the system. The IOC will wait for its bus grant signal, **gnt_l**, to be asserted and for the PCI to be idle before assuming bus ownership. The PCI is idle when both **frame_l** and **irdy_l** are sampled deasserted. In the next cycle the IOC will initiate a PCI transfer as described in Section 6.2.5.

The IOC will automatically deassert its **req_l** signal in the same cycle that it initiates the PCI bus cycle (coincident with **frame_l** assertion). If the IOC has another request queued and its grant is not asserted (i.e. the PCI bus is not parked with the CPU), it will not reassert **req_l** until one cycle after the completion of the current PCI transaction initiated by the IOC.

6.2.3 Default Mastership (Bus Parking)

When no devices are requesting bus mastership it is recommended that the system arbiter grant default bus ownership to the CPU by asserting the IOCs **gnt_l** signal. This will reduce the latency for CPU initiated PCI transfers when the bus is idle. Granting the PCI to a device when no requests are pending is referred to as 'parking' in the PCI Specification.

Should the IOC be granted the bus when it is not asserting it PCI request signal, **req_l**, the IOC will enable the **ad<31:00>**, **c_be_l<3:0>**, and **par** output drivers one cycle following the assertion of **gnt_l** to prevent them from floating. The IOC will tristate these output drivers one cycle following the deassertion of **gnt_l** (if it has not initiated a PCI transfer).

If the IOC's grant is asserted, the PCI is idle (both **frame_l** and **irdy_l** are sampled deasserted), and the IOC has a request queued, it will initiate a PCI transaction on the next PCI cycle without asserting **req_l**.

Preliminary

6.2.4 Memory Barrier Instruction Requirements

As described in Sections 2.4.3 and 2.4.3.1, the memory barrier (MB) instruction must be used to gaurantee strict write ordering and prevent write merging in the internal CPU write buffer. When executed, the MB instruction will force any data in the internal CPU write buffers to be flushed.

Because a shifted address is used when accessing PCI address spaces the behavior of the internal write buffer must be carefully considered. Writes to addresses on PCI that differ by at least a byte address will be written to different hexaword entries in the internal CPU write buffer. Writes to the same PCI address with different length encodings will result in writes to differnet qaudwords within the same hexaword write buffer entry. However, the IOC is designed to accept only a longword or quadword of data from any write buffer entry. Therefore, to insure proper operation writes to the same PCI address using different length encodings (byte, word, tribyte, unmasked) require the use of memory barrier instrucitons between each write. In addition, writes to the same PCI byte address and same length encoding will get merged in the internal CPU write buffer unless seperated by memory barrier instructions.

6.2.5 Address Phase

In the cycle immediately following the detection by the IOC of both a bus idle condition and **gnt_l** assertion, the IOC will initiate a PCI transaction by driving the PCI target address on **ad<31:0>**, a valid cycle code on **c_be_l<3:0>**, and asserting **frame_l**. The IOC will drive even parity data on **par** for the address in the cycle following **frame_l** assertion. All PCI transfers begin with this 'address phase'. If an address parity error is detected by any PCI device it will be reported on the PCI **serr_l** signal two cycles after the assertion of **frame_l**. Since the IOC does not directly connect to **serr_l** system logic can be used to report the error through an interrupt pin (see Section 6.2.13 for more detail).

Configuration cycles use address stepping which drives the address and command for two consecutive cycles. **frame_l** is asserted during the second of these cycles.

6.2.6 Sparse Memory Space

The sparse memory space comprises the region in which maskable CPU-initiated PCI transactions occur. As described above, this regions uses the low CPU address to indicate transfer size and uses the IOC_HAE register to extend the shifted address to cover the 32-bit PCI address space.

Preliminary

6.2.6.1 Generating PCI Addresses for Sparse Space

To support various transfer size (byte enable) combinations in sparse space the addresses used by the CPU to reference PCI address spaces are encoded. The encoding uses bits <4:3> of the physical address as size bits. Bits <31:7> of the physical address are used for PCI address bits <26:2>. Bits <6:3> of the physical address are used to generate the PCI byte enables **c_be_l<3:0>**, as shown in Table 6–2). The byte enables generated by combinations of physical address bits <6:3> not shown in Table 6–2) are unspecified. CPU physical address bits <2:0> must be zero for these transfers.

Table 6–2 Byte Enable Encoding for CPU Initiated Transfers in Sparse Space

CPU a<6:5>	CPU a<4:3>	PCI Byte Enables* c_be_l<3:0>	PCI ad<1:0> Memory Cycles	PCI ad<1:0> I/O Cycles	Transfer Type	Size
00	00	1110	00	00	Masked	Byte
01	00	1101	00	01	Masked	Byte
10	00	1011	00	10	Masked	Byte
11	00	0111	00	11	Masked	Byte
00	01	1100	00	00	Masked	Word
01	01	1001	00	01	Masked	Word
10	01	0011	00	10	Masked	Word
00	10	1000	00	00	Masked	Tribyte
01	10	0001	00	01	Masked	Tribyte
00	11	0000	00	00	Unmasked	Longword
11	11	0000	00†	00	Unmasked	Quadword

*Byte enables are asserted when zero.

†**ad2** is 0 for the first longword and 1 for the second longword of the quadword. The quadword is normally sent as a burst of two consecutive longwords, starting with the lower longword; in this case only one address (**ad2** = 0) is generated. If a retry on the second longword is needed, the address would be regenerated with **ad2** = 1.

During CPU initiated transfers to sparse memory space the IOC will force PCI address bits <1:0> to 00 which indicates to the target device that the transfer will use a linearly incrementing burst order.

Although the PCI supports arbitrary byte enables the IOC only supports the byte enable combinations listed in Table 6–2).

The following types of masked transfers are permitted in Sparse space:

- Longword reads: A LDL instruction can be used to generate a masked PCI read transfer. When the LDL instruction is used a PCI read transfer with a burst length of one will be generated.

Preliminary

- Longword writes: STL instruction must be used. A PCI write transfer with a burst length of one will be generated.

The following types of unmasked transfers are permitted in Sparse space:

- Unmasked longword reads: A LDL instruction can be used to generate an unmasked PCI read transfer, generating a burst length of one.
- Unmasked quadword reads: A LDQ instruction can be used to generate an unmasked PCI read transfer, generating a burst length of two.
- Unmasked longword writes: Unmasked PCI write transfers can be generated using the STL in struction. The IOC will generate a PCI write transfer with a burst length of one.
- Unmasked quadword writes: Unmasked PCI write transfers can be generated using the STQ instruction. A PCI write transfer with a burst length of two will be generated.

6.2.6.2 Address Extension For Sparse Space

Due to the encoding of the CPU address used to generate a PCI byte enables only 27 physical address bits (CPU address bits **a<31:5>**) can be used to generate PCI address bits **ad<26:0>**, giving an effective PCI address space of 128MB. This 128MB address space is subdivided into a 16MB region and a 112MB region. PCI address bits **ad<31:27>** are generated differently depending on which of these regions are referenced as described below and shown in Table 6–3.

Table 6–3 Host Address Extension

CPU a<33:32>	CPU a<31:29>	PCI ad<31:27>
10	000	00000
10	001	IOC_HAE<31:27>
10	010	IOC_HAE<31:27>
10	011	IOC_HAE<31:27>
10	100	IOC_HAE<31:27>
10	101	IOC_HAE<31:27>
10	110	IOC_HAE<31:27>
10	111	IOC_HAE<31:27>

The 16MB region is always mapped to the first 16MB of the PCI address space. This 16MB region is referenced when CPU address bits **a<31:29>** are zero. During CPU initiated PCI transactions to this 16MB region PCI address bits **ad<31:27>** will be zero.

Preliminary

The 112MB region is referenced when CPU address bits **a<31:29>** are nonzero. During CPU initiated PCI transactions to this 112MB region PCI address bits **ad<31:27>** will be generated using the host address extension (HAE) field in the IOC_HAE register (Section 6.5.1). The HAE field allows the 112MB region to be relocated to the upper 112MB of any aligned 128MB PCI address segment. A value of zero in the HAE field defines a single 128MB region, beginning at address 0 in PCI address space.

6.2.6.3 Burst Order in Sparse Space

For sparse memory space, bits <1:0> of the PCI address for memory cycles are used to specify the burst order being requested by the bus master. The microprocessor will always use a linear incrementing burst order. Only a CPU-initiated PCI unmasked-quadword read or write cycle can generate a transfer where the burst length is greater than 1 (Section 6.2.6.6). For dense space, bursts can be greater than one.

6.2.6.4 Unmasked Longword and Quadword Reads to Sparse Space

CPU-initiated PCI longword reads are generated by executing LDL or LDQ instructions that reference a physical address in the PCI sparse space. The encoded address used by the CPU must have address bits <6:5> equal to 00 for a longword and 11 for a quadword, and bits <4:3> equal to 11 as specified in Table 6–2. In the address phase of a CPU initiated PCI longword read cycle the IOC will drive a memory read command onto **c_be_l<3:0>**.

For a longword read, in the cycle following the address phase, the IOC will deassert the **frame_l** signal, assert the PCI initiator ready signal **irdy_l**, and begin sampling the target ready signal **trdy_l** and **ad<31:0>**. When **trdy_l** is sampled asserted by the IOC it will latch the read data on **ad<1:0>**. For a quadword read, the IOC will deassert the **frame_l** signal after the first as serted **trdy_l** is sampled.

For longword reads, in the next cycle the IOC will deassert its **irdy_l** signal and sample the parity data being driven on **par** by the target. The IOC will return the longword of read data from the PCI to the CPU on the low longword of the returned quadword.

For quadword reads, in the next cycle the IOC will latch the upper longword on **ad<1:0>** and parity associated with the first longword. Parity associated with the second longword will appear in the third cycle. The IOC will pack the two longwords and return a single quadword of read data to the CPU.

If the sampled parity data does not match the parity compute by the IOC then an error will get reported to the CPU by IOC via an internal interrupt (see Section 6.2.14.1 for more information). The deassertion of **irdy_l** signals the completion of the cycle to the PCI target device.

Preliminary

6.2.6.5 Unmasked Longword Writes to Sparse Space

CPU-initiated PCI longword writes are generated by executing STL instructions that reference a physical address in the PCI sparse memory. The encoded address used by the CPU must have address bits **a<6:5>** equal to 11 and **a<4:3>** equal to 11 as specified in Table 6–2. In the address phase of a CPU-initiated PCI longword write cycle the IOC will drive a memory write command onto **c_be_l<3:0>**.

In the cycle following the address phase, the IOC will deassert the **frame_l** signal, assert the PCI initiator ready signal **irdy_l**, drive valid write data onto **ad<1:0>**, and begin sampling the target ready signal **trdy_l**. In the next cycle the IOC will drive even parity data for **ad<1:0>** and **c_be_l<3:0>** on **par**. When **trdy_l** is sampled asserted by the IOC will deassert its **irdy_l** signal in the next PCI cycle which completes the write cycle. If the target of the write transaction detected a parity error it will assert the **perr_l** signal two cycles after the assertion of **trdy_l**. If the IOC samples **perr_l** asserted in this cycle (two cycles after the data transfer) it will report a parity error to the CPU via an internal interrupt (see Section 6.2.14.2 for more information).

The IOC will extract the longword of write data for the PCI from the longword of the CPU quadword indicated by the CPU physical address bit **a7** (0 = low, 1 = high) which corresponds to PCI address bit **ad2**.

6.2.6.6 Unmasked Quadword Writes to Sparse Space

CPU-initiated PCI quadword writes are generated by executing STQ instructions that reference a physical address in the PCI sparse memory. The encoded address used by the CPU must have address bits **a<6:5>** equal to 00 and **a<4:3>** equal to 11 as specified in Table 6–2.

In the address phase of a CPU-initiated PCI longword write cycle the IOC will drive a memory write command onto **c_be_l<3:0>**. To complete the quadword write the IOC initiates a two longword burst write cycle using linear incrementing burst order on the PCI.

In the cycle following the address phase, the IOC will continue to assert the **frame_l** signal, assert the PCI initiator ready signal **irdy_l**, drive the lower longword of write data onto **ad<1:0>**, and begin sampling the target ready signal **trdy_l**. When **trdy_l** is sampled asserted, the IOC will in the next PCI cycle drive the upper longword of write data onto **ad<1:0>**, deassert its **frame_l** signal and continue to assert its **irdy_l** signal. When **trdy_l** is again sampled asserted, the IOC will deassert its **irdy_l** signal in the next PCI cycle which completes the quadword write cycle.

Preliminary

If the target of the write transaction detected a parity error on either longword of data transferred it will assert the **perr_l** signal two cycles after the assertion of **trdy_l** corresponding to the data transfer for which the parity error was detected. If the IOC samples **perr_l** asserted two cycles after either data transfer it will report a parity error to the CPU via an internal interrupt (see Section 6.2.14.2 for more information).

The IOC will always transfer the low longword of the quadword write data first, followed by the high longword of the quadword write data.

6.2.6.7 Masked Longword Reads to Sparse Space

Masked PCI longword reads are generated by executing LDL instruction that references a physical address in the PCI sparse memory and have CPU address bits **a<4:3>** equal to 00, 01, or 10. The encoded address used by the CPU determines the byte mask used during the transfer as specified in Table 6–2. Masked reads proceed in a similar manner to unmasked longword reads. However, when the IOC is asserting **irdy_l** it is also asserting only the byte enables determined by the encoded address.

6.2.6.8 Masked Longword Writes to Sparse Space

Masked PCI longword writes are generated by executing STL instructions that reference a physical address in the PCI sparse memory and have CPU address bits **a<4:3>** equal to 00, 01, or 10. The encoded address used by the CPU determines the byte mask used during the transfer as specified in Table 6–2. Masked writes proceed in a similar manner to unmasked longword writes. However, when the IOC is asserting **irdy_l** it is also asserting only the byte enables determined by the encoded address.

6.2.7 Dense Memory Space

The Dense memory space comprises the region in which unmasked, longword or quadword transfers can be specified. The purpose of this space is to allow more efficient use of the on-chip write buffer by condensing transfers that span multiple quadwords into a single burst.

6.2.7.1 Generating PCI Addresses for Dense Space

This region uses the unshifted CPU address **a<31:0>** to generate the PCI address. Since there is adequate address space here, the IOC_HAE register is neither needed nor used to generate dense space PCI addresses. Note also that any PCI address can be generated from either dense or sparse space.

During CPU-initiated transfers to dense space the IOC will force PCI address bits **ad<1:0>** to 00 which indicates to the target device that the transfer will use a linearly incrementing burst order.

Preliminary

Because the address is not shifted, masked transfers are not permitted in dense space. Unmasked transfers are described in the following sections.

6.2.7.2 Unmasked Reads

The PCI will return an aligned quadword in response to a longword read regardless of the alignment of the address. The correct longword is returned to the register file in the low 32 bits. The PCI will return an aligned quadword in response to a quadword read. The address is assumed to be quadword-aligned.

6.2.7.3 Unmasked Burst Writes

Unmasked PCI write transfers up to one hexaword can be generated by a combination of STL and STQ instructions which have addresses falling within an aligned hexaword. It is not necessary that all longwords of this hexaword be written; that is, the softwae can write any longword portion of the hexaword. The IOC will simply drive sequence through an unwritten longword by deasserting the byte enables for that longword.

The IOC uses a linear incrementing burst order on the PCI to sequence through successive longwords.

In the cycle following the address phase, the IOC will continue to assert the **frame_l** signal, assert the PCI initiator ready signal **irdy_l**, drive the first longword of write data onto **ad<1:0>**, and begin sampling the target ready signal **trdy_l**. In the PCI cycle after **trdy_l** is sampled asserted, the IOC will drive the next longword of write data onto **ad<1:0>**, deassert its **frame_l** signal and continue to assert its **irdy_l** signal. This will continue until the last longword. When **trdy_l** is again sampled asserted, the IOC will deassert its **irdy_l** signal in the next PCI cycle which completes the burst.

If the target of the write transaction detected a parity error on either longword of data transferred it will assert the **perr_l** signal two cycles after the assertion of **trdy_l** corresponding to the data transfer for which the parity error was detected. If the IOC samples **perr_l** asserted two cycles after either data transfer it will report a parity error to the CPU via an internal interrupt (see Section 6.2.14.2 for more information).

The IOC will always transfer the low longword of the quadword write data first, followed by the high longword of the quadword write data.

Preliminary

6.2.8 I/O Space Cycles

The IOC generates addresses in this region in order to specify commands to the PCI-architected I/O region. This region supports identical access types and masked address generation as Sparse space (see Section 6.2.6) with the following exceptions.

- I/O Space is accessed with CPU **a<33:32>** = 01 and CPU **a<31:29>** = 110
- The IOC_HAE register is not used; instead, PCI **ad<1:27>** = 00000
- PCI address bits **ad<1:0>** will come directly from physical address bits **a<6:5>** during CPU initiated PCI I/O transfers.
- An I/O read or write command will be issued on **c_be_l<3:0>** corresponding to a read or write to this region.

The region supports all masked and unmasked transfers that are supported in sparse space.

6.2.9 Configuration Cycles

The PCI provides for a configuration address space in order to initialize and configure devices via configuration registers defined in this space. The IOC allows access to these registers by supporting transfers to configuration address space. This region supports identical access types and masked address generation as sparse space (see Section 6.2.6) with the following exceptions.

- Configuration space is accessed with CPU **a<33:32>** = 01 and CPU **a<31:29>** = 111
- The IOC_HAE register is not used; instead, PCI **ad<1:27>** = 00000
- PCI address bits **ad<1:0>** will come directly from configuration cycle type register (IOC_CONF, Section 6.5.2) bits <1:0> during CPU initiated PCI configuration space transfers.
- A configuration space read or write command will be issued on **c_be_l<3:0>** corresponding to a read or write to this region.

It is important to note that during configuration cycles, addresses will be restricted by the encoding requirements of PCI configuration cycles as described in the PCI Local Bus Specification, Revision 2.0 and also by the system-dependent implementation of PCI initialization device select (IDSEL) encoding. During configuration cycles, address stepping of one cycle occurs.

Preliminary

6.2.10 Special Cycles and Interrupt Acknowledge Cycles

The PCI provides a special cycle command to implement a simple broadcast mechanism. This allows processor status informantion and sideband signaling. The command is not addressed to a specific device and the receivers determine whether the message is applicable to it. See the PCI Local Bus Specification, Revision 2.0 for more information.

The CPU can initiate a special cycle transfer on the PCI by executing a STL instruction to the interrupt vector and special cycle register (IOC_IACK_SC, Section 6.5.11). When this register is written, the IOC will initiate a PCI cycle with a special cycle bus command. The address generated on the PCI by the IOC during a special cycle is unspecified and ignored by PCI devices. The write data generated by the STL instruction will pass unmodified through the IOC to the PCI during the data phase of the special cycle. The special cycle message is encoded in the write data as described in the PCI Local Bus Specification, Revision 2.0.

The special cycle address region is accessed by STL only. It supports identical write access types and masked address generation as sparse space (see Section 6.2.6) with the following exceptions.

- The IOC_IACK_SC register is write-accessed with CPU **a<33:32>** = 01 and CPU **a<31:29>** = 101
- The IOC_HAE register is used to generate PCI **ad<1:27>** under all circumstances.
- A special cycle command will be issued on **c_be_l<3:0>** and a PCI interrupt acknowledge cycle will ensue.

The PCI provides for an interrupt acknowledge cycle in which the IOC can acknowledge an interrupt and receive an interrupt vector. This allows external interrupt arbitration hardware to supply interrupt vectors on behalf of PCI devices.

The CPU can initiate an interrupt acknowledge cycle by executing either a LDL or LDQ instruction, depending on the size of the returned interrupt vector. Read access types and mask generation is identical to sparse space (see Section 6.2.6) with the following exceptions:

- The IOC_IACK_SC register is read-accessed with CPU **a<33:32>** = 01 and CPU **a<31:29>** = 101
- The IOC_HAE register is used to generate PCI **ad<1:27>** under all circumstances.

Preliminary

- An interrupt acknowledge cycle command will be issued on **c_be_l<3:0>** and a PCI interrupt acknowledge cycle will ensue.

It is recommended that software will use only address 1A000 0000 to reference the IOC_IACK_SC register provided by the IOC. However, in the current chip implementation, CPU address bits <28:0> are not decoded during an access of the IOC_IACK_SC register and any address in the range 1Axxxxxxx to 1Bxxxxxxx will alias to this single register. For convenience of implementation the chip will drive the supplied address onto the PCI.

When the IOC_IACK_SC register is read, the IOC will initiate a PCI cycle with an interrupt acknowledge bus command. Interrupt arbitration logic external to chip is needed to supply an interrupt vector in response to the interrupt acknowledge bus cycle. The interrupt vector data will be returned to the CPU as the IOC_IACK_SC register read data. The address generated by the IOC during an interrupt acknowledge cycle is unspecified and is ignored by PCI devices.

During CPU initiated interrupt acknowledge cycles PCI address bits **ad<31:0>** generated by the IOC are unspecified.

6.2.11 Burst Order

For sparse memory space, PCI address bits **ad<1:0>** for memory cycles are used to specify the burst order being requested by the bus master. The microprocessor will always use a linear incrementing burst order. Only a CPU-initiated PCI unmasked quadword read or write cycle can generate a transfer where the burst length is greater than one (see Section 6.2.6.6). For dense space, bursts can be greater than one.

6.2.12 Back-to-Back Transfers

The IOC always inserts a bus idle (bus turnaround) cycle following the completion of a CPU initiated PCI transfer. If the IOC is still requesting the PCI and its grant signal is sampled asserted in the idle cycle the IOC will initiate another transfer in the next PCI cycle. This sequence of two uninterrupted transfers by the IOC is termed a back to back transfer by the PCI specification. Note that due to the idle cycle between back to back transfers the IOC is not capable of PCI fast back to back transfers.

Preliminary

6.2.13 PCI Address Parity

For CPU initiated PCI transactions the IOC will calculate even parity on the address driven on **ad<31:0>** and the cycle command code driven on **c_be_l<3:0>** and drive the result on the PCI **par** signal pin in the next PCI clock cycle. Because the IOC is the source of the address, command code, and parity data the IOC will not detect or report address parity errors during CPU initiated PCI transactions. PCI peripherals that detect PCI address parity errors will report them via the PCI **serr_l** signal. Since chip does not provide an **serr_l** signal pin external logic is expected to report **serr_l** assertions via one of the chip's interrupt signal pins.

6.2.14 PCI Data Parity

For CPU initiated PCI write transfers the IOC will generate even parity for write data. The PCI **ad<31:0>** and **c_be_l<3:0>** signals are included in the parity computation. The IOC will check for even parity for read data during CPU initiated PCI read transfer. Parity is generated/checked for all bytes of the PCI bus (independent of the byte enables).

6.2.14.1 PCI Read Data Parity Errors

During a CPU initiated PCI read data transfer the IOC will compute even parity for the **ad<31:0>** and C_BE<3:0>_L sampled by the IOC. The computed parity will be compared with the value received on **par** in the PCI clock cycle following a valid data transfer (both **irdy_l** and **trdy_l** are asserted. If they do not compare a read data parity error has been detected. The IOC will report the error to the CPU via an internal interrupt and log the error condition as described below. The read data returned to the CPU will also be flagged with an error witch will result in an instruction exception.

If the ERR bit in the IOC status 0 register (IOC_STAT0, Section 6.5.6) is not already set, the IOC will set the ERR bit and set the error CODE field in the IOC_STAT0 register to 2 (bad data parity). The PCI command code of the CPU initiated transfer on which the error was detected will be logged in the CMD field of the IOC_STAT0 register. The CMD bit field will indicate that the error occurred during a PCI read cycle. The starting address of the CPU initiated PCI transfer will also be logged in the ADDR field of the IOC status 1 register (IOC_STAT1, Section 6.5.7). The ERR bit and the logged error information will be frozen until the ERR bit is cleared by writing a one to the ERR bit in the IOC_STAT0 register.

If the ERR bit in the IOC_STAT0 register is already set, the IOC will set the LOST error bit in the IOC_STAT0 register. The LOST bit will remain set until cleared by writing a one to the LOST bit in the IOC_STAT0 register.

Preliminary

6.2.14.2 PCI Write Data Parity Errors

During a CPU initiated PCI write data transfer the IOC will compute even parity for the **ad<31:0>** and C_BE<3:0>_L driven by the IOC. The computed parity will be driven on **par** in the PCI clock cycle after the write data is driven. PCI devices that support PCI parity checking will compute even parity for the **ad<31:0>** and C_BE<3:0>_L it receives when they are the target of a PCI memory, I/O or configuration write cycles. The target samples **par** one cycle after a valid data transfer (both **irdy_l** and **trdy_l** are asserted). If the computed parity does not compare to the parity received on **par** a write data parity error has been detected and the target will assert **perr_l** in the next cycle (two cycles after a valid data transfer). The IOC will report the error to the CPU via an internal interrupt and log the error condition as described in Section 6.2.14.1. The CMD bit field logged in the IOC_STAT0 register will indicate that the error was generated during a PCI write cycle.

6.2.15 PCI Master Timeout

The PCI protocol specifies a mechanism to limit the duration of a masters burst sequence. The mechanism requires PCI masters to implement a latency timer that counts the number of cycles since the assertion of FRAME#. If the master latency timer has expired and the masters grant has been deasserted the master is required to surrender the bus. This mechanism is intended to prevent masters from holding bus ownership for extended periods of time and trades off high throughput for low latency. Since the IOC has a maximum burst length of two with no initiator delay between transfers it does not implement the master latency timer.

6.2.16 PCI Target Disconnects

All PCI bus masters must support the target disconnect cycle terminations. Target disconnects are signaled through the use of the PCI signal **stop_l** as described in the PCI specification. When the IOC detects a target disconnect termination being signaled by the target it will end the current transfer. If the target disconnect was not a target-abort, and the disconnected transfer was not completed, the IOC will attempt to complete the transfer by rearbitrating for PCI mastership and initiating a transfer starting at the address of the next untransferred data. An error will not be reported unless a target-abort disconnect was detected (see Section 6.2.18) or the PCI retry limit is exceeded (see Section 6.2.17).

Preliminary

6.2.17 PCI Retry Limit

If the PCI target of a CPU initiated transfer retries the transfer 2^24 times a retry timeout error will be generated and the IOC will delete the CPU initiated transfer from its queue. If the CPU initated transaction was a read then undefined data will be returned to the CPU with error status which will result in an instruction exception. The IOC will also report an error to the CPU via an interrupt and log the error condition as described below.

If the ERR bit in the IOC_STAT0 register is not already set, the IOC will set the ERR bit and set the error CODE field in the IOC_STAT0 register to 0 (retry limit). The PCI command code of the CPU initiated transfer on which the error was detected will be logged in the CMD field of the IOC_STAT0 register. The CMD bit field will indicate the PCI cycle type in which the error occurred. The starting address of the CPU initiated PCI transfer will also be logged in the ADDR field of the IOC_STAT1 register. The ERR bit and the logged error information will be frozen until the ERR bit is cleared by writing a one to the ERR bit in the IOC_STAT0 register.

If the ERR bit in the IOC_STAT0 register is already set, the IOC will set the LOST error bit in the IOC_STAT0 register. The LOST bit will remain set until cleared by writing a one to the LOST bit in the IOC_STAT0 register.

6.2.18 PCI Target Abort

The target abort termination can be signaled by a PCI target during a transaction in which a fatal error has occurred. A PCI target may also signal a target abort termination when the target will never be able to respond to a transaction. Upon detecting a target abort termination the IOC will end the transaction by deasserting **frame_l** and asserting **irdy_l**, and then deasserting **irdy_l** one cycle later. The IOC will not retry this access on the PCI. If the CPU initiated access was a read the IOC will return arbitrary data with error status to the CPU which will result in an instruction exception. The IOC will also report an error to the CPU via an interrupt and log the error condition as described below.

If the ERR bit in the IOC_STAT0 register is not already set, the IOC will set the ERR bit and set the error CODE field in the IOC_STAT0 register to 3 (target abort). The PCI command code of the CPU initiated transfer on which the error was detected will be logged in the CMD field of the IOC_STAT0 register. The CMD bit field will indicate the PCI cycle type in which the error occurred. The starting address of the CPU initiated PCI transfer will also be logged in the ADDR field of the IOC_STAT1 register. The ERR bit and the logged error information will be frozen until the ERR bit is cleared by writing a one to the ERR bit in the IOC_STAT0 register.

Preliminary

If the ERR bit in the IOC_STAT0 register is already set, the IOC will set the LOST error bit in the IOC_STAT0 register. The LOST bit will remain set until cleared by writing a one to the LOST bit in the IOC_STAT0 register.

6.2.19 PCI Master Abort

If no PCI device has responded by asserting **devsel_l** to a CPU initiated PCI transfer within five PCI clock cycles after the assertion of **frame_l** the IOC will terminate the PCI transaction. If the CPU initiated access was a read the IOC will return arbitrary data with error status to the CPU which will result in an instruction exception. The IOC will also report the error to the CPU via an interrupt and log the error condition as described below. The IOC will not retry this transfer on the PCI.

If the ERR bit in the IOC_STAT0 register is not already set, the IOC will set the ERR bit and set the error CODE field in the IOC_STAT0 register to 1 (no device). The PCI command code of the CPU initiated transfer on which the error was detected will be logged in the CMD field of the IOC_STAT0 register. The CMD bit field will indicate the PCI cycle type in which the error occurred. The address of the CPU initiated PCI transfer will also be logged in the ADDR field of the IOC_STAT1 register. The ERR bit and the logged error information will be frozen until the ERR bit is cleared by writing a one to the ERR bit in the IOC_STAT0 register.

If the ERR bit in the IOC_STAT0 register is already set, the IOC will set the LOST error bit in the IOC_STAT0 register. The LOST bit will remain set until cleared by writing a one to the LOST bit in the IOC_STAT0 register.

6.3 Peripheral Initiated PCI Cycles

The IOC functions as the bridge between PCI and the system memory and CPU. The IOC functions as a PCI target (slave) when responding to cycles initiated by a PCI peripheral device or external DMA controller operating as a bus master. PCI initiated masked reads are treated as unmasked reads by the IOC. Arbitrary byte enable combinations for PCI initiated write transfers are implemented by read-modify-write operations in the memory controller. PCI initiated transfers can target pages of memory that are mapped in cacheable or noncacheable address spaces. The chip is designed so that memory coherency is maintained for all PCI initiated transfers to cacheable memory.

Registers internal to chip are not accessible by PCI devices.

Preliminary

6.3.1 PCI Cycle Command Code Support and Aliasing

The IOC supports only PCI initiated memory transfers. PCI memory read and memory write cycle command types are supported directly. Other PCI memory command types are supported by aliasing them to either the memory read or memory write command as shown in Table 6–4 below.

Table 6–4 Supported Commands for PCI Initiated Transactions

c_be_l<3:0>	PCI Command Type	Comments
0110	Memory Read	Directly supported
0111	Memory Write	Directly supported
1100	Memory Read Multiple	Aliased to Memory Read command
1110	Memory Read Line	Aliased to Memory Read command
1111	Memory Write and Invalidate	Aliased to Memory Write command

The IOC will not respond to PCI initiated transfers with the cycle command codes listed in Table 6–5 below.

Table 6–5 Unsupported Commands for PCI Initiated Transfers

c_be_l<3:0>	PCI Command Type
0000	Interrupt Acknowledge
0001	Special Cycle
0010	I/O Read
0011	I/O Write
0100	Reserved *
0101	Reserved*
1000	Reserved*
1001	Reserved*
1010	Configuration Read
1011	Configuration Write
1101	Dual Address Cycle

*Reserved command types are reserved in the PCI specification.

6.3.2 PCI Address Translation

Since PCI uses a 32 bit address and the CPU uses a 34 bit address, PCI addresses that the IOC responds to must be translated to an equivalent address in the CPUs address space. The IOC provides two programmable address windows that control access by PCI peripherals to system memory. These address windows are referred to as the PCI Target Windows. There is a set of three IOC registers associated with each PCI Target Window referred

Preliminary

to as the Window Base register, Window Mask register, and Translated Base register.

The Window Mask register provides a mask corresponding to bits <31:20> of an incoming PCI address. The size of each window can be programmed to be from 1 Mbyte to 4 Gbytes in powers of two by masking bits of the incoming PCI address via the Window Mask register as shown in Figure 6–2.

Note

Figure 6–2 shows only the supported values of W_MASK<31:20>. Other values are unsupported.

Figure 6–2 PCI Window Mask

W_MASK<31:20>	Window Size
0000 0000 0000	1MB
0000 0000 0001	2MB
0000 0000 0011	4MB
0000 0000 0111	8MB
0000 0000 1111	16MB
0000 0001 1111	32MB
0000 0011 1111	64MB
0000 0111 1111	128MB
0000 1111 1111	256MB
0001 1111 1111	512MB
0011 1111 1111	1GB
0111 1111 1111	2GB
1111 1111 1111	4GB

Bits <31:20> of the Window Base register specify a starting address in the PCI memory address space for the target window. The Window Base and Window Mask registers are used as described below to determine when a PCI transaction references an address in a Target Window. The window base address must be on a naturally aligned address boundary equal to the window size (see Figure 6–2).

Preliminary

Bits <31:20> of incoming PCI addresses are bitwise XORed with bits <31:20> of each window's Window Base register. The result of each window's bitwise XOR is then bitwise ANDed with the complement of bits <31:20> of that window's Window Mask register. The result of each window's bitwise AND is then ORed to produce a hit (match) or miss indication. If the result of the final OR is a zero for a given window, then the incoming address has hit in that PCI Target Window, otherwise it has missed in that window. A window enable bit, W_EN, is provided in each windows Window Base register to allow windows to be independently enabled or disabled. If a windows W_EN bit is set, then the window is enabled. PCI initiated transfers that hit in the window will be responded to by the IOC. If the W_EN bit is cleared then the window is disabled and PCI initiated transfers that hit in the window will be ignored by the IOC.

If a hit occurs in any PCI Target Window that is enabled, then the IOC will respond to the PCI cycle by asserting **devsel_l**. The PCI Target Windows must be programmed so that the PCI address ranges that they respond to do not overlap. This insures that only one PCI Target Window can match an incoming PCI address at any one time.

When an address match occurs with a PCI Target Window, the IOC translates the 32 bit PCI address to a 34 bit CPU address. The translated address is generated in one of two ways as determined by the SG (Scatter Gather) bit of the window's Window Base register. If the SG bit is cleared then direct mapped address translation is used (see Section 6.3.2.1). If the SG bit is set then scatter gather mapped address translation is used (see Section 6.3.2.2).

6.3.2.1 Direct Mapped Address Translation

When an address match occurs in a PCI Target Window during an incoming PCI transaction and the SG bit in the corresponding Window Base register is cleared the incoming PCI address is translated to an address in the CPUs memory address space using via a direct mapped translation. The translated address is generated by concatenating bits from the matching window's Translated Base register with bits from the incoming PCI address. Bits <33:32> of the translated address will always be 00. The Window Mask register determines which bits of the Translated Base register and PCI address are used to generate the translated address as show in Figure 6–3. The resultant address is used as the start ing address in system memory for the PCI transaction.

Preliminary

Caution

Unused bits of the translated base register must be cleared (zero) for correct operation.

Figure 6–3 PCI Address Translation - Scatter Gather Mapping Disabled

W_MASK <31:20>	Translated Address <31:0> T_BASE	PCI Address	Unused T_BASE Bits
0000 0000 0000	<31:20>	<19:0>	<19:9>
0000 0000 0001	<31:21>	<20:0>	<20:9>
0000 0000 0011	<31:22>	<21:0>	<21:9>
0000 0000 0111	<31:23>	<22:0>	<22:9>
0000 0000 1111	<31:24>	<23:0>	<23:9>
0000 0001 1111	<31:25>	<24:0>	<24:9>
0000 0011 1111	<31:26>	<25:0>	<25:9>
0000 0111 1111	<31:27>	<26:0>	<26:9>
0000 1111 1111	<31:28>	<27:0>	<27:9>
0001 1111 1111	<31:29>	<28:0>	<28:9>
0011 1111 1111	<31:30>	<29:0>	<29:9>
0111 1111 1111	<31:31>	<30:0>	<30:9>
1111 1111 1111	<31:32>	<31:0>	<31:9>

6.3.2.2 Scatter Gather Mapped Address Translation

When an address match occurs in a PCI Target Window during an incoming PCI transaction and the SG bit in the corresponding Window Base register is set the incoming PCI address is translated to an address in the CPU's memory address space using via a scatter gather mapped translation. The translated address is generated by a table lookup. The incoming PCI address is used to index a table stored in system memory. This table is referred to as a Scatter Gather Map. The IOC generates the index address into the Scatter Gather Map by combining the incoming PCI address with the Translated Base register of the matching PCI Target Window. The Translated Base register specifies the starting address of the Scatter Gather map table. Bits of the incoming PCI address are used as an offset from the base of the table.

The length of the Scatter Gather Map Table is determined by the size of the PCI Target Window as defined by the Window Mask register as shown in Figure 6–4. Each Scatter Gather Map entry maps an 8K page of PCI address space into an 8K page of the CPUs address space. Each Scatter Gather Map entry is a quadword (64 bits) so the low three bits of a Scatter Gather Map address are always zero. The Window Mask register determines which bits of

Preliminary

the Translated Base register and PCI address are used to generate the address of the Scatter Gather Map entry as shown in Figure 6–4. Bits <33:32> of the translated address are always 00. The resultant address is used to read an entry from the Scatter Gather Map table stored in system memory. Note that the unused bits of the Translated Base register as indicated in Figure 6–4 must be cleared for proper operation.

Caution

Unused bits of the translated base register must be cleared (zero) for correct operation.

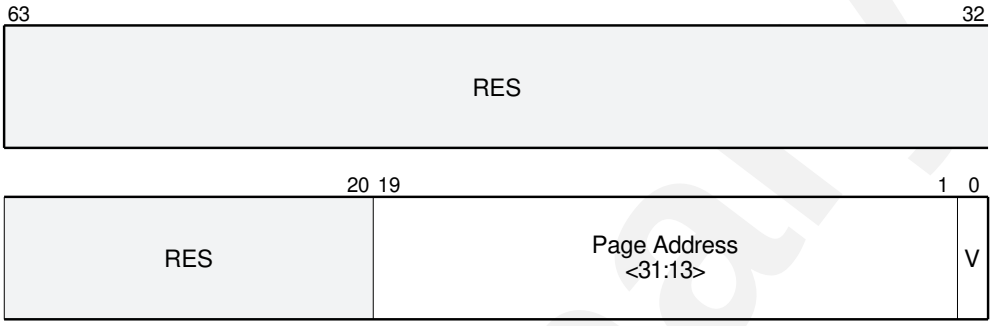
Figure 6–4 Scatter Gather Map Address

W_MASK <31:20>	Scatter–Gather Map Table Size	Translated Address <31:0> T_BASE	PCI Address		Unused T_BASE Bits
0000 0000 0000	1KB	<31:10>	<19:13>	000	None
0000 0000 0001	2KB	<31:11>	<20:13>	000	<10>
0000 0000 0011	4KB	<31:12>	<21:13>	000	<11:10>
0000 0000 0111	8KB	<31:13>	<22:13>	000	<12:10>
0000 0000 1111	16KB	<31:14>	<23:13>	000	<13:10>
0000 0001 1111	32KB	<31:15>	<24:13>	000	<14:10>
0000 0011 1111	64KB	<31:16>	<25:13>	000	<15:10>
0000 0111 1111	128KB	<31:17>	<26:13>	000	<16:10>
0000 1111 1111	256KB	<31:18>	<27:13>	000	<17:10>
0001 1111 1111	512KB	<31:19>	<28:13>	000	<18:10>
0011 1111 1111	1MB	<31:20>	<29:13>	000	<19:10>
0111 1111 1111	2MB	<31:21>	<30:13>	000	<20:10>
1111 1111 1111	4MB	<31:22>	<31:13>	000	<21:10>

The Scatter Gather Map entry read from system memory specifies a page address in the CPUs 34 bit address space. Since the page size is 8 Kbytes, and bits <33:32> of the translated address are always 00, only address bits <31:13> are specified by the page table entry. The quadword page table entry is formatted as shown in Figure 6–5. The Scatter Gather Map entry includes an valid bit (V). If this bit is cleared, the Scatter Gather Map entry is invalid. When the IOC encounters an invalid entry during a translation an error condition will be logged and the error will be reported to the CPU via an interrupt and the IOC will also terminate the PCI transfer (see Section 6.3.5).

Preliminary

Figure 6–5 Scatter Gather Map Page Table Entry



The page address specified by Scatter Gather Map entry is then used to translate the incoming 32 bit PCI address to an equivalent 34 bit CPU address. This is the final translation of the Scatter Gather map process. The resulting 34 bit translated address is the address in the CPUs address space of the memory location being referenced by the incoming PCI address.

The final translated address is generated by concatenating the page address (bits <31:13>) read from the Scatter Gather Map table with the low bits of the incoming PCI address(bits <12:0>). Address bits <33:32> will always be 00. This 34 bit translated address is the physical address used by the IOC to access a system memory location to complete the peripheral initiated PCI cycle. If the translated address is not decoded by the Memory Controller an error condition will be logged, the error will be reported to the CPU via an interrupt, and the IOC will terminate the PCI transfer using the Target Abort protocol (see Section 6.3.8.2).

6.3.3 Scatter Gather Translation Buffer

An eight entry Translation Look-aside Buffer (TLB) is provided in the IOC for Scatter Gather Map entries. The TLB is a fully associative cache. Each entry in the TLB stores a PCI page address (cache tag) and the corresponding CPU page address (cache data) from eight most recent Scatter Gather Map lookups. Each time an incoming PCI address selects a PCI Target Window that has Scatter Gather Mapping enabled, bits <31:13> of the PCI address are compared with the PCI page addresses in the TLB(tag). If a match is found, the CPU page address is provided by the data of that matching TLB entry. The Scatter Gather Map Table lookup is avoided resulting in higher performance (lower latency).

Preliminary

If no match is found in the TLB, the Scatter Gather Map lookup is performed as described earlier. The data read from the Scatter Gather Map Table and the incoming PCI page address are written over an existing entry in the TLB. The TLB entry replaced is determined according to a round robin algorithm.

All entries in the TLB can be flushed by writting to the IOC Translation Buffer Invalidate All register (IOC_TBIA). When the IOC_TBIA reigster has been written all entries in the TLB will be invalidated at the start of the next PCI transaction. The TLB will be also flushed if a Page Table read error occurs.

If the TEN bit in the TB enable register (IOC_TB_ENA, Section 6.5.3) is cleared all entries will be invalidated at the start of every subsequent PCI bus cycle (during the address decoding perfromed by the IOC). Clearing the TEN bit does not disable scatter gather mapped translations. If a scatter gather map translation is required for a PCI initiated transaction while the TEN bit is cleared the IOC will read the required TLB entry from system memory and temporarily store the entry in the TLB for the current transaction. This entry will be invalidated at the start of any subsequent PCI transation if TEN is still cleared.

6.3.4 Page Table Read Errors

If the address of the scatter gather map entry references a nonexistent address, the read data has an uncorrectable error, or a cache tag parity error is detected when reading cached data, the IOC will report the error to the CPU via an internal interrupt and log error information as described below. If the PCI initiated transfer was a read then the IOC will terminate the PCI transaction using the target abort protocol. If the PCI initiated transfer was a write then the IOC completes the bus cycle normally but does not transfer the write data to memory.

If the ERR bit was not set prior to this transaction bits <31:13> of the starting address of the incoming PCI transfer are logged in the P_NBR field of the IOC status 0 register (IOC_STAT0, Section 6.5.6). The ERR bit is set to indicate that a page table read error occurred. In addition the error CODE field in the IOC_STAT0 register is set to 5 (page table read error). If the ERR bit was set prior to this transaction the LOST bit in the IOC_STAT0 register will be set. The LOST bit will remain set until cleared by writing a one to the LOST bit in the IOC_STAT0 register.

Preliminary

6.3.5 Invalid Page Errors

If the page table entry read from system memory is invalid the IOC will report an error to the CPU via an internal interrupt and log error information as described below. If the PCI initiated transfer was a read then the IOC will terminate the PCI transaction using the target abort protocol. If the PCI-initiated transfer was a write then the IOC completes the bus cycle normally but does not transfer the write data to memory.

If the ERR bit was not set prior to this transaction bits <31:13> of the starting address of the incoming PCI transfer are logged in the P_NBR field of the IOC_STAT0 register. The ERR bit is set to indicate that a page table read error occurred. In addition the error CODE field in the IOC_STAT0 register is set to 6 (invalid page). If the ERR bit was set prior to this transaction the LOST bit in the IOC_STAT0 register will be set. The LOST bit will remain set until cleared by writing a one to the LOST bit in the IOC_STAT0 register.

6.3.6 Address Phase

When the IOC detects the assertion of **frame_l** by an external PCI master it sample the address (**ad<31:0>**), and PCI command code (**c_be_l<3:0>**). During the next PCI clock cycle the IOC will decode the address and determine if the reference was to an address in a PCI Target Window, and will sample the parity information for the address phase from **par**. If the address hit in a PCI Target Window, the IOC will assert **devsel_l** in the next PCI clock cycle (two cycles after **frame_l** assertion) signaling to the PCI master that the IOC will respond to the PCI transaction.

6.3.6.1 PCI Address Queue

The IOC has a two entry address queue for PCI initiated transactions. When the IOC samples an address it inserts the address and command code in its PCI address queue. The PCI address queue can hold address and command information for up to two PCI initiated transactions. When a transaction completes the associated address queue entry is cleared (free). Because the IOC buffers write data for PCI initiated write transactions it is possible that a write transaction will complete on PCI while there is still write data to be transferred to memory held in the IOCs PCI write data queue. In that case, the address and command information for a subsequent PCI initiated transaction will get stored in the second entry of the IOCs PCI address queue. The second transaction will be stalled (**trdy_l** will not be asserted) until the buffered write transaction completes.

Preliminary

6.3.6.2 Address Burst Order

The IOC supports only linear incrementing burst order. If bits <1:0> of an incoming PCI address are 00 then linear incrementing burst order has been specified by the initiator of the PCI transaction. The IOC will allow a burst to continue until a page boundary is crossed (see Section 6.3.6.3), the target latency time-out has been exceeded (see Section 6.3.9), or an error occurs.

If bits <1:0> of an incoming PCI address are anything other than 00 then a burst order other than linear incrementing order has been specified by the initiator of the PCI transaction. In this case, the IOC will complete only one data transfer. If the initiator requests a transfer with a burst length greater than one, the IOC will terminate the transfer using the PCI disconnect protocol. This forces the initiator to break the burst transfer into multiple single transfer cycles. Each of these single transfer cycles will specify a new address that is consistent with the burst order desired by the initiator.

6.3.6.3 Page Boundary Crossing

The IOC will stop a PCI write burst that crosses an 8 Kb page boundary using the Target Disconnect protocol. This will force PCI devices to break transfers at 8 Kb page boundaries. No errors will be generated when an 8 Kb page boundary is reached. The IOC will not assert the PCI **stop_l** signal until the cycle after it asserts **trdy_l** to complete the transfer of the last longword address within the current 8 Kb page.

6.3.6.4 Address Parity Errors

If an address parity error is detected by the IOC during a PCI-initiated transaction IOC will report the error to the CPU via an internal interrupt and log the error condition as described below. The IOC will complete the transaction as normal on PCI. If the PCI transaction was a read transfer then the IOC will return the requested read data from memory. However, if the PCI transaction was a write transfer then the IOC will not write the data to system memory (the data will be lost).

If the ERR bit was not set prior to this transaction the starting address of the PCI transfer will be logged in the ADDR field of the IOC status 1 register (IOC_STAT1, Section 6.5.7). The ERR bit is set to indicate that a page table read error occurred. In addition the error CODE field in the IOC_STAT0 register is set to 4 (address parity error). The PCI command code of the incoming PCI initiated transfer on which the error was detected will be logged in the CMD field of the IOC_STAT0 register. If the ERR bit was set prior to this transaction the LOST bit in the IOC_STAT0 register will be set. The LOST bit will remain set until cleared by writing a one to the LOST bit in the IOC_STAT0 register.

Preliminary

6.3.7 PCI Initiated Write Data Transfers

When the IOC receives a PCI initiated write transfer, it transfers the write data received from the PCI bus to system memory via the memory controller. Because the chip uses a quadword memory organization, read-modify-write cycles must be used when less than a full quadword is being written.

If the first longword received for a given PCI initiated bus cycle is an odd longword address, the memory controller will perform a read-modify-write operation on the destination address to implement the longword write.

If the IOC receives a longword of write data for an even longword address that is not the last longword of the bus cycle, the IOC waits until it receives the next longword before issuing a write to the destination address. In this case, the IOC packs both longwords into a single quadword and writes the data to the destination address with a single quadword write transfer. If either longword is masked, the memory controller performs a read-modify-write operation on the addressed location.

If the last longword received is on an even longword address, the memory controller uses a read-modify-write operation to complete the longword transfer.

6.3.7.1 PCI Write Data Queue

The IOC implements a four quadword entry PCI write data queue for incoming data during a PCI initiated memory write cycle. The PCI write data queue is not used during PCI initiated memory read cycles. If a PCI initiated write burst transfer begins on a odd longword address boundary the first entry in the write data queue will store only one longword of data (in the upper longword of the queue entry). The IOC will pack two longwords of write data into each of entry of the PCI write data queue if the burst begins on a quadword aligned boundary or after the first data transfer of an odd longword aligned transfer. If the burst ends on an even longword address then the last queue entry will store only one longword of data (in its low longword).

As each entry of the PCI write data queue is successfully written to system memory it is used to store subsequent write burst data.

6.3.7.2 PCI Initiated Memory Write Data Errors

Nonexistent memory errors or uncorrectable read errors that occur during a read-modify-write memory operation resulting from a PCI-initiated memory write transfer are not reported by the IOC. These errors are reported to the CPU by the memory controller.

Preliminary

During a PCI initiated memory write data transfer the IOC will compute even parity for **ad<31:0>** and **c_be_l<3:0>** sampled by the IOC. The computed parity will be compared with the value received on **par** in the next PCI clock cycle. If they do not compare a write data parity error has been detected. The IOC will assert **perr_l** during the next PCI clock cycle to inform the initiator that a write data parity error has been detected.

The data with the parity error will be written to memory. The IOC will report the error to the CPU via an internal interrupt and log the error condition as described below.

If the ERR bit was not set prior to this transaction the starting address of the PCI transfer will be logged in the ADDR field of the IOC_STAT1 register. The error CODE field in the IOC_STAT0 register is set to 7 (data error). The PCI command code of the incoming PCI initiated transfer on which the error was detected will be logged in the CMD field of the IOC_STAT0 register. The CMD bit field will indicate that the error occurred during a PCI write cycle. If the ERR bit was set prior to this transaction the LOST bit in the IOC_STAT0 register will be set. The LOST bit will remain set until cleared by writing a one to the LOST bit in the IOC_STAT0 register.

6.3.8 PCI Initiated Read Data Transfers

To increase performance for PCI initiated burst read transfers the IOC will prefetch read data. Once the address of a PCI initiated read is loaded into the PCI address queue the IOC will begin to prefetch quadwords from sequential memory locations. If the master attempts to continue a burst across an 8 Kb page boundary the IOC will terminate the transaction using the target disconnect protocol when the last longword in the 8 Kb page is transfer. When the IOCs prefetch address counter reaches the last longword of an 8 Kb boundary it will wrap back to the starting address of the same 8 Kb page. Even though the counter has wrapped the IOC will continue to prefetch data. However, this prefetch data will be flushed from the IOCs prefetch queue when the PCI read transaction is terminated via the disconnect protocol by the IOC.

As the memory read data is returned to the IOC the data is stored in the PCI read data queue (see Section 6.3.8.1) along with the associated error status for the read data. The IOC returns the requested longwords of read data to the PCI initiator from the read data queue.

Since the IOC prefetches read data, the IOC may still be waiting for remaining prefetch data to be returned from memory when the PCI initiator completes its transfer. If another PCI transaction is responded to by the IOC while the prefetch reads resulting from the prior transaction are still pending, the IOC will buffer the new address in the PCI request queue. However, the IOC

Preliminary

will not issue any new memory requests until all the prefetch data requested during the previous transaction is received and flushed from the read queue.

6.3.8.1 PCI Read Data Queue

The PCI read data queue contains four entries for the read data of PCI initiated read transfers. The queue allows the IOC to prefetch read-data as discussed in Section 6.3.8. Each entry contains a quadword aligned quadword. The queue is only used for one bus cycle at a time and is flushed upon the completion of the current PCI initiated read transfer.

6.3.8.2 PCI Initiated Memory Read Data Errors

Nonexistent memory errors or uncorrectable read errors that occur during a PCI initiated read transfer are reported by the IOC as described below.

Errors that occur during a prefetch read are not immediately reported. When prefetch read data is taken from the read queue and driven onto the PCI the associated error status bit is checked. If the error bit is set the IOC will terminate the PCI read transfer using the target abort protocol. The IOC will also report the error to the CPU via an internal interrupt and log the error as described below.

If the ERR bit was not set prior to this transaction the starting address of the PCI transfer will be logged in the ADDR field of the IOC_STAT1 register. The error CODE field in the IOC_STAT0 register is set to 7 (data error). The PCI command code of the incoming PCI initiated transfer on which the error was detected will be logged in the CMD field of the IOC_STAT0 register. The CMD bit field will indicate that the error occurred during a PCI read cycle. If the ERR bit was set prior to this transaction the LOST bit in the IOC_STAT0 register will be set. The LOST bit will remain set until cleared by writing a one to the LOST bit in the IOC_STAT0 register.

6.3.9 PCI Target Latency Timeout

If the latency of a memory operation introduces more than eight PCI cycles of latency since the last **trdy_l** assertion by the IOC for a PCI initiated memory transfer then the IOC will terminate the transfer using the disconnect protocol by asserting **stop_l** in the next PCI cycle. It is possible that data will be transferred in the bus cycle in which the IOC signals a disconnect. This will occur if the master is asserting **irdy_l** and the IOC state changes such that it asserts **trdy_l** in the same cycle that it asserts **stop_l**.

Preliminary

6.3.10 PCI Initiated Exclusive Access

The IOC supports the PCI exclusive access protocol for PCI initiated transfers that access system memory via the IOC and memory controller. The IOC will consider an access to be an exclusive access and mark itself locked when the following sequence of events occur and the stated conditions are met during a PCI transfer.

1. **lock_l** is sampled deasserted in the same cycle that **frame_l** asserts.
2. The PCI initiated cycle addresses a location in an enabled PCI Target Window.
3. **lock_l** is sampled asserted by the IOC in a cycle that is subsequent to the assertion of **frame_l**, but prior to the cycle in which the IOC asserts **trdy_l**.

While the IOC is locked the internal CPU lock flag will be held reset preventing the completion of store conditional instructions. The IOC will maintain the locked status until it samples both **frame_l** and **lock_l** deasserted.

While the IOC is locked it will continue to sample **lock_l** in the same cycle that **frame_l** asserts during subsequent PCI initiated transfers. If **lock_l** is sampled asserted the transfer is a non-exclusive access. If this non-exclusive cycle addresses a location in an enabled PCI Target Window the IOC will use the PCI disconnect protocol to signal a retry and no data will be transferred. If **lock_l** is sampled deasserted the transfer is a continuation of the exclusive access. If the address for the exclusive access cycle a location in an enabled PCI Target Window complete the access normally (the IOC will not retry the access).

6.3.11 Guaranteed Access Arbitration

In some applications it may be necessary for a PCI device to win arbitration to memory before initiating a PCI cycle so that access latency can be minimized. The IOC supports two PCI sideband signals, **memreq_l** and **memack_l**, for this purpose. Both sideband signals are synchronous to the PCI clock signal (**pci_clk_in**). When a device needs to win arbitration to memory prior to initiating a PCI cycle it must assert **memreq_l** to the IOC. When the IOC samples **memreq_l** asserted it will arbitrate internally to the chip for access to memory. The IOC will assert **memack_l** when arbitration to memory has been won. While **memack_l** is asserted the CPU will be blocked from accessing memory. Once the requesting device samples **memack_l** it can initiate a PCI cycle (provided it has won arbitration for PCI) and be assured that an intervening CPU access will not introduce additional latency. **memack_l** will remain asserted until one cycle after **memreq_l** is removed.

Preliminary

6.4 CPU Access of IOC Registers

The IOC provides a number of registers that control its operation or aid in its testability. These registers can be accessed directly by the CPU but are not accessible from the PCI. Detailed descriptions of the registers and their bit fields appear in Section 6.5.

6.5 IOC Registers

This section describes each of the IOC registers, the address at which the CPU can access them, and their bit assignments. All references to IOC registers should be quadword accesses. Longword access of the IOC registers is not supported. All bits of all IOC registers are not modified by the assertion of RESET_L and will be undefined after a power up reset except where noted.

The R/W (read/write) column of the register bit descriptions indicates the access behavior of the bit(s) as listed in Table 6–6 below.

Table 6–6 IOC Register Bit Access Behavior

Type	Behavior
WO	Write-only, UNPREDICTABLE on reads
RO	Read-only, ignored on writes
R/W1C	Read; write a value of 1 to clear
X	UNPREDICTABLE on reads, ignored on writes

6.5.1 Host Address Extension Register (IOC_HAE)

This register is accessed at address 1 8000 0000. All bits of this register are undefined after reset.

Preliminary

Figure 6–6 Host Address Extension Register (IOC_HAE)

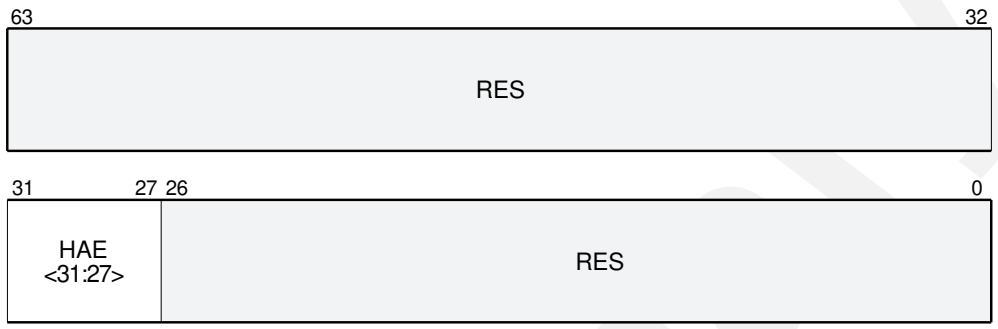


Table 6–7 Host Address Extension Register (IOC_HAE) Bit Description

Bits	Field	Type	Description
26:0	RES	X	Reserved
31:27	HAE<31:27>	WO	The HAE<31:27> bit field will be used for PCI address bits **ad<31:27>** during CPU initiated PCI transactions where the physical CPU address bits <31:29> are nonzero.
63:32	RES	X	Reserved

6.5.2 Configuration Cycle Type Register (IOC_CONF)

This register is accessed at address 1 8000 0020. All bits of this register are undefined after reset.

Preliminary

Figure 6–7 Configuration Cycle Type Register (IOC_CONF)

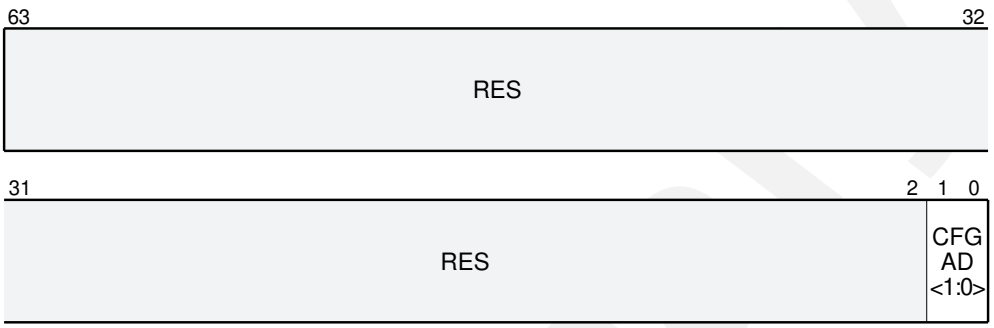


Table 6–8 Configuration Cycle Type Register (IOC_CONF) Bit Description

Bits	Field	Type	Description
1:0	CFG AD<1:0>	WO	The CFG_AD<1:0> bit field will be used as PCI address bits **ad<1:0>** during CPU initiate PCI configuration cycles.
63:2	RES	X	Reserved

6.5.3 Translation Buffer Enable Register (IOC_TB_ENA)

This register is accessed at address 1 8000 00A0. All bits of this register are undefined after reset.

Figure 6–8 Translation Buffer Enable Register (IOC_TB_ENA)

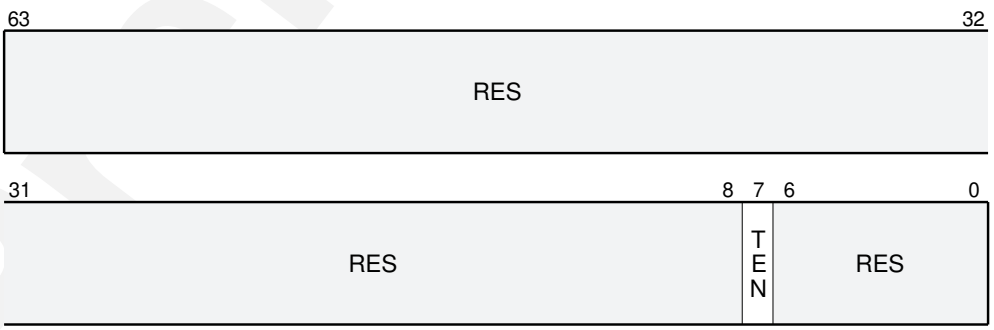


Table 6–9 Translation Buffer Enable Register (IOC_TB_ENA) Bit Description

Bits	Field	Type	Description
6:0	RES	X	Reserved
7	TEN	WO	When the T_EN bit is written with a zero the scatter gather map translation buffer (TLB) is disabled. The TLB will remain disabled until the T_EN bit is written with a one. Disabling the TLB does not disable scatter gather map translations.
63:8	RES	X	Reserved

6.5.4 PCI Soft Reset Register (IOC_SFT_RST)

The PCI Soft Reset Register is accessed at address 1 8000 00C0. The RST bit will be set after reset.

Figure 6–9 PCI Soft Reset Register (IOC_SFT_RST)

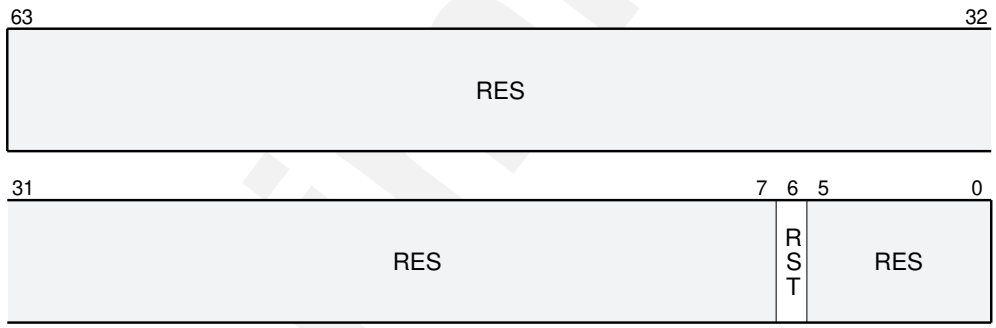


Table 6–10 PCI Soft Reset Register (IOC_SFT_RST)Bit Description

Bits	Field	Type	Description
5:0	RES	X	Reserved

(continued on next page)

Preliminary

Table 6–10 (Cont.) PCI Soft Reset Register (IOC_SFT_RST)Bit Description

Bits	Field	Type	Description
6	RST	WO	When the RST bit is written with a one the PCI **rst_l** signal will be asserted by the IOC. **rst_l** will remain asserted until the RST bit is written with a zero. The required assertion time of **rst_l** will be system dependent and must be timed by software. Note that the RST bit will be set and the PCI **rst_l** signal will be asserted when the chip reset signal, **reset_l**, is asserted.
63:7	RES	X	Reserved

6.5.5 PCI Parity Disable Register (IOC_PAR_DIS)

This register allows parity checking and reporting to be disabled. It is accessed at address 1 8000 00E0. The PAR bit is cleared after reset.

Figure 6–10 PCI Parity Disable Register (IOC_PAR_DIS)

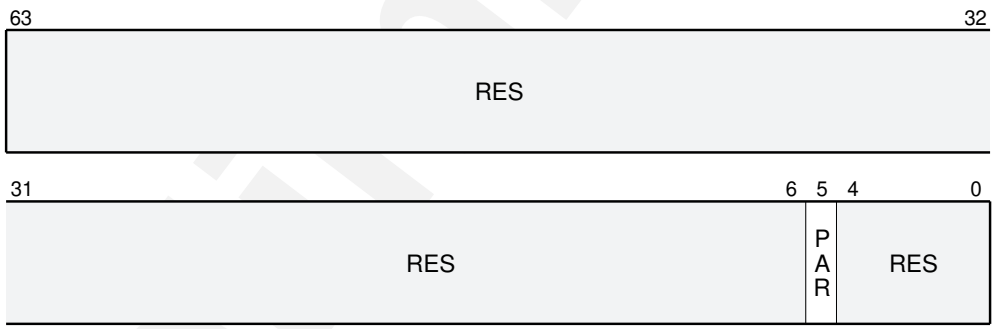


Table 6–11 PCI Parity Disable Register (IOC_PAR_DIS) Bit Description

Bits	Field	Type	Description
4:0	RES	X	Reserved

(continued on next page)

Preliminary

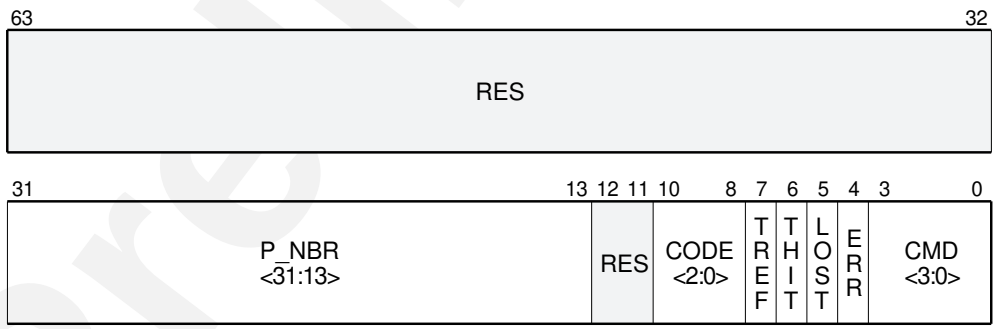
Table 6–11 (Cont.) PCI Parity Disable Register (IOC_PAR_DIS) Bit Description

Bits	Field	Type	Description
5	PAR	WO	When this bit is set, parity checking by the IOC (comparing the parity bit to computed parity) is disabled and parity errors detected in this manner are neither logged nor reported. For CPU writes to the PCI, the target PCI device will report parity errors on **perr_l** and an error is logged (bad data parity, CODE = 2) in the IOC_STAT0 register. This reporting is not masked by the **par** bit. This bit is initialized to zero.
63:6	RES	X	Reserved

6.5.6 Status 0 Register (IOC_STAT0)

The IOC Status 0 register is accessed at address 1 8000 0040. This register is used by the IOC to log information pertaining to an error condition. If the ERR bit is not set and an IOC error is detected, all bits of the IOC_STAT0 register will be updated and the ERR bit will be set. All bits of the IOC_STAT0 register except the LOST, T_REF, T_HIT bits will be frozen until the ERR bit is cleared by software. All bits of this register are undefined after reset. The LOST bit will be set when an IOC error is detected and the ERR bit is already set. The LOST bit will remain set until it is cleared by software.

Figure 6–11 Status 0 Register (IOC_STAT0)



Preliminary

Table 6–12 Status 0 Register (IOC_STAT0) Bit Description

Bits	Field	Type	Description
3:0	CMD <3:0>	RO	The CMD bit field stores the PCI command of a CPU initiated PCI transaction or a PCI initiated transaction during which an error, other than a page table read error or invalid page table error, was detected by the IOC.
4	ERR	R /W1C	This bit is set when an error has been logged by the IOC. This bit is cleared when written with a value of 1.
5	LOST	R /W1C	This bit is set when an error occurs and either the ERR bit is already set or the CODE is 5 or 6 and the ERR bit was set at the beginning of the transaction which caused the error. This signifies that error logging information pertaining to the new error has been lost. This bit is cleared when written with a value of 1.
6	THIT	RO	T_HIT is provided for diagnostic test usage. T_HIT will be set if the last PCI transaction that referenced an address in a Target Window also hit in the IOCs address translation buffer (TLB). T_HIT will be cleared if the last PCI transaction that referenced an address in a Target Window also missed in the IOCs address translation buffer (TLB).
7	TREF	RO	T_REF is provided for diagnostic test usage. T_REF will be set if the last PCI transaction referenced an address in a Target Window. T_REF will be cleared if the last PCI transaction referenced an address outside of the address ranges defined by the Target Windows.

(continued on next page)

Preliminary

Table 6–12 (Cont.) Status 0 Register (IOC_STAT0) Bit Description

Bits	Field	Type	Description
10:8	CODE <2:0>	RO	The CODE<2:0> bit field contains the error code for the error condition that caused the ERR bit to set. The encoding of the CODE bit field is as follows:

Code	Error Condition
CPU Initiated Transfers	
000	Retry limit
001	No device
010	Bad data parity
011	Target abort
PCI Initiated Transfers	
100	Bad address parity
101	Page table read error
110	Invalid page
111	Data error

12:11	RES	X	Reserved
31:13	P_NBR <31:13>	RO	The P_NBR bit field stores address bits **ad<31:13>** of the starting PCI address for a PCI initiated transfer during which a page table read error or invalid page table error was detected.
63:32	RES	X	Reserved

6.5.7 Status 1 Register (IOC_STAT1)

The IOC Status 1 register is accessed at address 1 8000 0060. All bits of this register are undefined after reset.

Preliminary

Figure 6–12 Status 1 Register (IOC_STAT1)

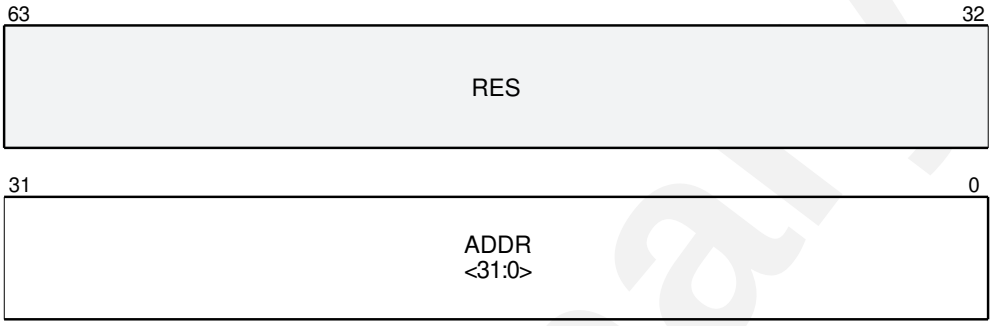


Table 6–13 Status 1 Register (IOC_STAT1) Bit Description

Bits	Field	Type	Description
31:0	ADDR<31:0>	RO	The ADDR bit field stores the starting address **ad<31:0>**, of a CPU initiated PCI transaction or a PCI initiated transaction during which an error, other than a page table read error or invalid page table error, was detected by the IOC.
63:32	RES	X	Reserved

6.5.8 Translation Buffer Invalidate All Register (IOC_TBIA)

The IOC Translation Buffer Invalidate All register is accessed at address 1 8000 0080. When the IOC_TBIA register is written all entries in the scatter gather translation buffer will be invalidated at the start of the next PCI initiated transaction. All bits of this register ignore writes and are undefined during reads.

Preliminary

Figure 6–13 Translation Buffer Invalidate All Register (IOC_TBIA)

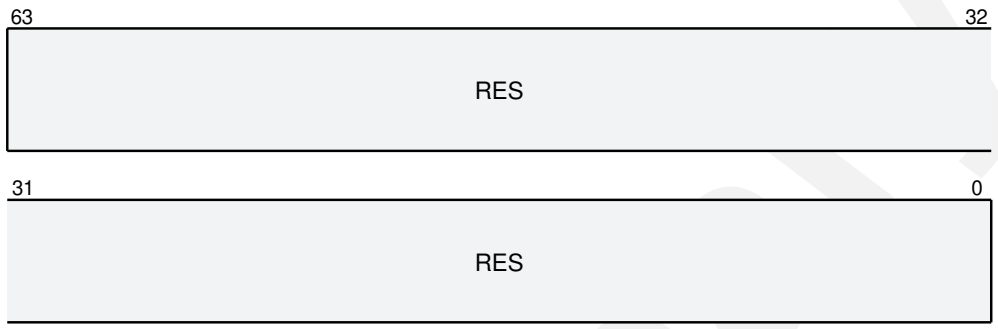


Table 6–14 Translation Buffer Invalidate All Register (IOC_TBIA) Bit Description

Bits	Field	Type	Description
63:0	RES	X	Reserved

6.5.9 PCI Target Window Registers

There are three registers, Window Base Register (W_BASE),Window Mask Register (W_MASK), and Translated Base Register (T_BASE), per PCI Target Window. Two sets of these registers, one per PCI Target Window, are provided by the IOC. The addresses of each of these registers are listed in the table below. Descriptions of the format for each register follow the table.

Table 6–15 PCI Target Window Register Addresses

Register	Address
W_BASE_0	1 8000 0100
W_BASE_1	1 8000 0120
W_MASK_0	1 8000 0140
W_MASK_1	1 8000 0160
T_BASE_0	1 8000 0180
T_BASE_1	1 8000 01A0

Preliminary

6.5.9.1 Window Base Registers (IOC_W_BASE<1:0>)

There are two Window Base registers: IOC_W_BASE0 and IOC_W_BASE1. The Window Base registers should not be modified unless software insures that the PCI is idle during the update of the register. All bits of these registers are undefined after reset.

Figure 6–14 Window Base Registers (IOC_W_BASE<1:0>)

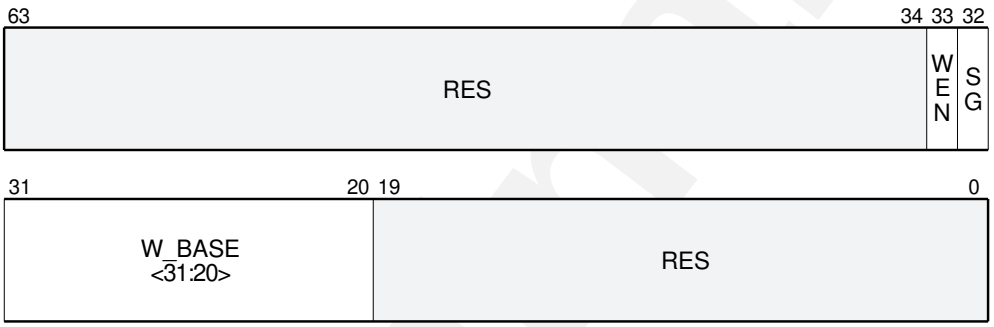


Table 6–16 Window Base Registers (IOC_W_BASE<1:0>) Bit Description

Bits	Field	Type	Description
19:0	RES	X	Reserved
31:20	W_BASE <31:20>	WO	W_BASE specifies the PCI base address of the PCI Target Window.
32	SG	WO	When the SG bit is cleared, the PCI Target Window uses direct mapping to translate a PCI address to a CPU address. When the SG bit is set, the PCI Target Window uses scatter gather mapping to translate a PCI address to a CPU address.
33	WEN	WO	When W_EN is cleared, the PCI Target Window is disabled and will not respond to PCI initiated transfers. When W_EN is set, the PCI Target Window is enabled and will respond to PCI initiated transfers that hit in the address range of the Target Window.
63:34	RES	X	Reserved

Preliminary

6.5.9.2 Window Mask Registers (IOC_W_MASK<1:0>)

There are two Window Mask registers: IOC_W_MASK0 and IOC_W_MASK1. The Window mask registers should not be modified unless software insures that the PCI is idle during the update of the register. All bits of these registers are undefined after reset.

Figure 6–15 Window Mask Registers (IOC_W_MASK<1:0>)

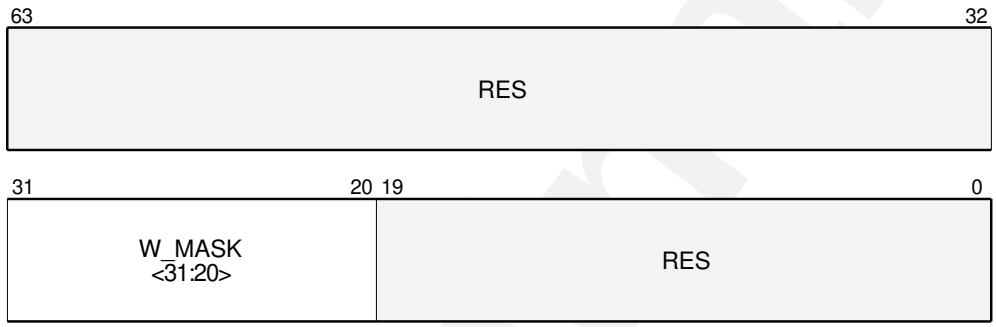


Table 6–17 Window Mask Registers (IOC_W_MASK<1:0>) Bit Description

Bits	Field	Type	Description
19:0	RES	X	Reserved
31:20	W_MASK <31:20>	WO	W_MASK specifies the size of the PCI Target Window and is also used in the translation of the PCI address to the CPU address to mask address bits not used to determine an address match to the PCI Target Window.
63:32	RES	X	Reserved

6.5.9.3 Translated Base Registers (IOC_T_BASE<1:0>)

There are two translated base registers: IOC_T_BASE0 and IOC_T_BASE1. The translated base registers should not be modified unless software insures that the PCI is idle during the update of the register. All bits of these registers are undefined after reset.

Preliminary

Figure 6–16 Translated Base Registers (IOC_T_BASE<1:0>)

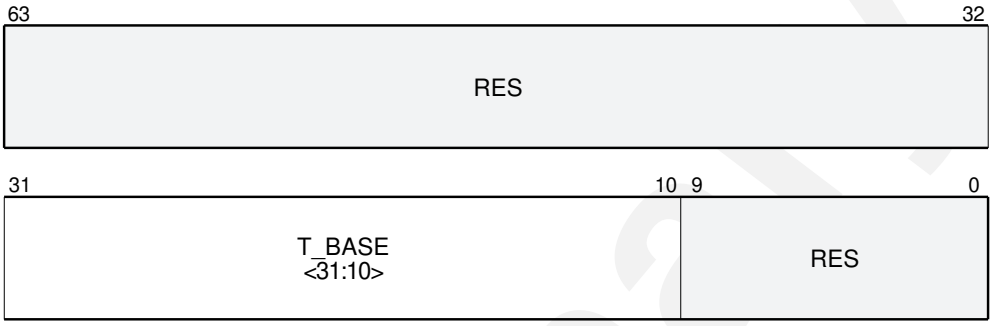


Table 6–18 Translated Base Registers (IOC_T_BASE<1:0>) Bit Description

Bits	Field	Type	Description
9:0	RES	X	Reserved
31:10	T_BASE <31:10>	WO	If Scatter Gather Mapping is disabled T_BASE specifies the base CPU address of the translated PCI address for the PCI Target Window. If Scatter Gather Mapping is enabled T_BASE specifies the base CPU address for the Scatter Gather Map Table for the PCI Target Window.
63:32	RES	X	Reserved

6.5.10 Translation Buffer Tag Registers (IOC_TB_TAG<7:0>)

Eight translation buffer tag registers, one for each translation buffer entry, are provided by the IOC. The addresses of each register is listed in Table 6–19.

Table 6–19 TLB Tag Register Addresses

Register	Address
IOC_TB_TAG0	1 8100 0000
IOC_TB_TAG1	1 8100 0020
IOC_TB_TAG2	1 8100 0040
IOC_TB_TAG3	1 8100 0060
IOC_TB_TAG4	1 8100 0080
IOC_TB_TAG5	1 8100 00A0
IOC_TB_TAG6	1 8100 00C0

(continued on next page)

Preliminary

Table 6–19 (Cont.) TLB Tag Register Addresses

Register	Address
IOC_TB_TAG7	1 8100 00E0

The IOC translation buffer tag registers are automatically updated by the IOC when the TEN bit in the IOC_TB_ENA register is set, a PCI transaction hits in a PCI Target Window, and the W_EN and SG bits in the IOC_W_BASE register associated with the PCI Target Window are both set.

The IOC provides the ability to directly write the translation buffer tag registers directly for use by diagnostic software. All bits of the IOC_TB_TAG<7:0> registers are undefined after reset.

Figure 6–17 Translation Buffer Tag Registers (IOC_TB_TAG<7:0>)

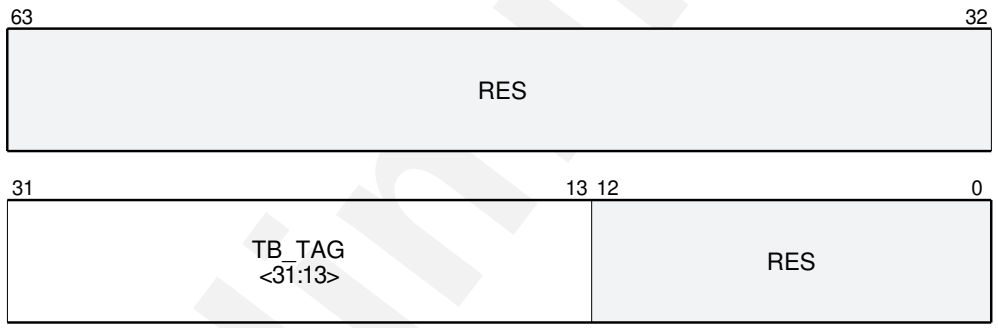


Table 6–20 Translation Buffer Tag Registers (IOC_TB_TAG<7:0>) Bit Description

Bits	Field	Type	Description
12:0	RES	X	Reserved
31:13	TB_TAG <31:13>	WO	This field provides Write-only access to one of eight TB tags.
63:32	RES	X	Reserved

Preliminary

6.5.11 Interrupt Vector and Special Cycle Register (IOC_IACK_SC)

This register is accessed at any quadword aligned address in the 1A0000000 to 1BFFFFFE0 address range.

When this register is read the IOC will generate an interrupt acknowledge cycle on PCI and return the vector data provided by external logic as the read data to the CPU. When this register is written the IOC will generate a PCI transfer with a special cycle command code. The data written to the register will be transferred to the PCI unmodified and contains the special message data.

All bits of this register are undefined after reset.

Figure 6–18 Interrupt Vector/Special Cycle Register (IOC_IACK_SC)

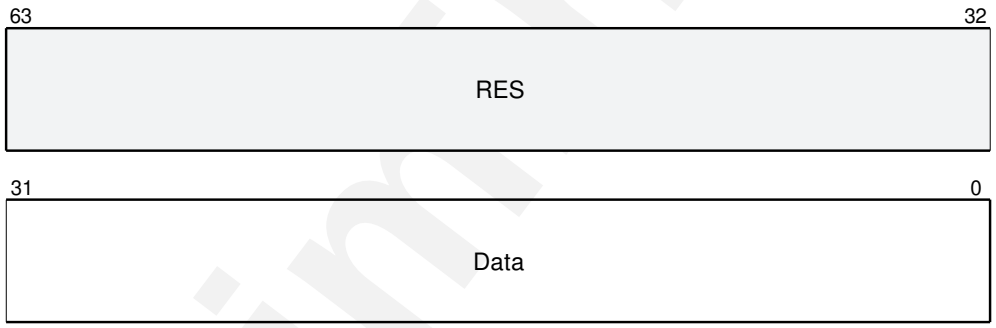


Table 6–21 Interrupt Vector/Special Cycle Register (IOC_IACK_SC) Bit Description

Bits	Field	Type	Description
31:0	DATA	RW	When the IOC_IACK_SC register is read a PCI interrupt acknowledge cycle will be generated by the IOC and the data returned will be an interrupt vector supplied by external system specific logic. When the IOC_IACK_SC is written the IOC will generate a PCI special cycle and the data written to this register will be used as the special cycle command (data).
63:32	RES	X	Reserved

Preliminary

6.5.12 Initialization Requirements

After a system power up or reset condition (RESET_IN_L is asserted) the following IOC register bits must be initialized by serial ROM code. This initialization is adequate to allow CPU initiated PCI transfers to operate correctly. Additional initialization will be required by boot code to configure the IOC to respond to PCI initiated transfers (DMA).

1. Write zero to the IOC_SFT_RST register to deassert reset.
2. Write ones to IOC_STAT0 bits <5:4> to clear the error and lost bits.
3. Write zero to W_BASE<1:0> registers to prevent the IOC from responding to itself.
4. Initialize the IOC_HAE register.

6.6 IOC Signal Pins

The signal pins used by the IOC are described in the following sections.

6.6.1 ad<31:0>

PCI address and data are multiplexed on the **ad<31:0>** pins. During the first clock of a PCI transaction the byte address is driven on the **ad<31:0>** pins. During subsequent clock cycles, **ad<31:0>** are driven with data.

ad<31:0> are bidirectional, tristateable signals.

6.6.2 c_be_l<3:0>

PCI bus command codes and byte enables are multiplexed on the **c_be_l<3:0>** pins. During the address cycle of a PCI transaction, the PCI bus command code is driven onto **c_be_l<3:0>**. During data cycles, inverted byte enables are driven on **c_be_l<3:0>**.

c_be_l<3:0> are bidirectional, tristateable signals.

6.6.3 frame_l

frame_l is asserted at the beginning of a PCI transaction. **frame_l** is also used to control the number of data transfers during the transaction (burst length). **frame_l** is deasserted during the final data phase of a transaction.

frame_l is a bidirectional, tristateable signal.

Preliminary

6.6.4 trdy_l

trdy_l is asserted by the target of a PCI transaction to signal the selected devices ability to complete the current data phase of a PCI transaction. During a read cycle the assertion of **trdy_l** indicates that valid data is being driven onto **ad<31:0>** by the selected device. During a write cycle the assertion of **trdy_l** indicates that the selected device is ready to accept write data. The current data phase completes when both **trdy_l** and **irdy_l** are sampled asserted.

trdy_l is a bidirectional, tristateable signal.

6.6.5 irdy_l

irdy_l is asserted by the initiator of a PCI transaction to signal its ability to complete the current data phase of a PCI transaction. During a read cycle the assertion of **irdy_l** indicates that the initiator is ready to accept read data. During a write cycle the assertion of **trdy_l** indicates that the initiator is driving valid write data onto **ad<31:0>**. The current data phase completes when both **trdy_l** and **irdy_l** are sampled asserted.

irdy_l is a bidirectional, tristateable signal.

6.6.6 stop_l

stop_l is driven by the target of a PCI transaction to request that the initiator stop the current transaction.

stop_l is a bidirectional, tristateable signal.

6.6.7 par

par is the even parity data bit for **ad<31:0>** and **c_be_l<3:0>**.

par is a bidirectional, tristateable signal.

6.6.8 perr_l

perr_l is asserted when a data parity error has been detected.

perr_l is a bidirectional, tristateable signal.

6.6.9 devsel_l

devsel_l is asserted by the device that is addressed by the current PCI transaction.

devsel_l is a bidirectional, tristateable signal.

Preliminary

6.6.10 lock_l

lock_l is used to implement atomic (exclusive) access operations on PCI.

lock_l is an input-only signal.

In test mode, this pin is also used to test the retry time out counter at the chip tester. Refer to the pin descriptions in Appendix A Table A-2.

6.6.11 req_l

req_l is asserted when the CPU accesses needs to initiate a PCI transfer. It is expected that external arbitration logic will be provided.

req_l is an output-only, tristateable signal.

6.6.12 gnt_l

gnt_l is asserted by external arbitration logic when the CPU is granted ownership of the PCI. It is expected that the arbiter will be designed to park (default grant) the PCI to the CPU when no other device is requesting ownership.

During configuration cycles (reads or writes), the IOC will perform an address stepping (drives address and command for 2 PCI cycles). If **gnt_l** is deasserted after the IOC has driven the address and command for 1 cycle but before **frame_l** is asserted, then IOC will relinquish the bus and wait for the **gnt_l** assertion and the PCI bus idle condition. It is possible that the microprocessor will never be able to complete a configuration cycle if minimum assertion time for **gnt_l** is one PCI cycle. The system designer must design around this potential problem by giving the microprocessor adequate time to complete its transactions. A simple solution is to design a 2-cycle minimum assertion time for **gnt_l**.

gnt_l is an input-only signal.

6.6.13 rst_l

rst_l is the PCI reset signal and is generated by the CPU. The RST bit in the IOC_SFT_RST register (Section 6.5.4) allows **rst_l** to be asserted under software control. **rst_l** is automatically asserted when the RESET_IN_L pin is asserted.

rst_l is an output-only signal.

Preliminary

6.6.14 pci_clk_in

pci_clk_in provides timing for all transactions on PCI. All of the IOCs PCI signals except **rst_l** are synchronous to **pci_clk_in**. Inputs are sampled on the rising edge of **pci_clk_in**. Outputs change state as a result of the rising edge of **pci_clk_in**.

pci_clk_in is an input-only signal.

6.6.15 memreq_l

memreq_l is a PCI sideband input-only signal and is synchronous to **pci_clk_in**. When the IOC samples **memreq_l** asserted it will arbitrate internally to the chip for access to memory. The IOC will assert **memack_l** when arbitration to memory has been won.

6.6.16 memack_l

memack_l is a PCI sideband output-only signal and is synchronous to **pci_clk_in**. When the IOC has won arbitration internal to the chip for access to memory as requested by **memreq_l** it will assert **memack_l**. **memack_l** will remain asserted until **memreq_l** is deasserted.

Preliminary

Preliminary

7
Clocks

This chapter describes the pin interface to the clock section and gives an overview of the PLL system to explain powerup frequency selection and noise reduction requirements.

7.1 Introduction

The chip employs a Phase Locked Loop (PLL) to synthesize a higher frequency internal clock from a lower frequency external clock.

Note that the synthesized clock is only used internally and has no timing relationship to PCI bus activity. No timing relationship between the external reference clock, the internal clock or test clock output should be assumed.

7.2 Clock Pins

The clock interface pins are summarized in Appendix A. The function of each clock pin is described below.

7.2.1 pll_clk_in, pll_clk_in_l

To minimize jitter induced by module and package noise, **pll_clk_in** and **pll_clk_in_l** receive a high-speed (> 50MHz) differential reference clock (logically complementary nominal square waves). For normal operation, a lower-speed (< 50 MHz) single-ended clock and an appropriate reference/bias voltage are supplied to the **pll_clk_in** and **pll_clk_in_l**, respectively.

Caution

pll_clk_in must be active before power is applied to the chip with **pll_bypass** asserted.

Preliminary

7.2.2 pll_bypass

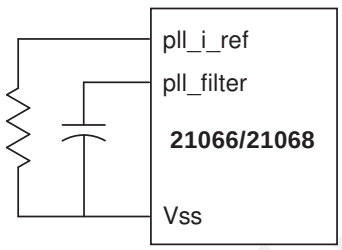
When **pll_bypass** is asserted high, the external clock input, **pll_clk_in**, drives the internal logic directly. The frequency ratio of the internal clock to the external reference will be 1:1.

pll_bypass should change state only when **reset_in_l** is asserted.

7.2.3 pll_i_ref

The constant current that flows in the [TBD] Ohm resistor connected between **pll_i_ref** and VSS provides the reference to analog PLL circuits in order to reduce the variations in the speed of CMOS devices over a wide range of process and operating conditions. Figure 7–1 shows the setup.

Figure 7–1 Reference Resistor and Filter Capacitor



7.2.4 pll_filter

A [TBD] µF capacitor connected between **pll_filter** and VSS regulates the speed with which the PLL responds to changes in frequency or operating conditions by setting the proper feedback loop time constant for maintaining stable operation.

7.2.5 pll_5v

The "clean" +5V supplied to **pll_5v** is regulated internally to source the nominal +3.3V used by the PLL and associated logic. This on-chip isolation is necessary to reduce phase jitter. 5V decoupling capacitors should be connected as close to the actual **pll_5v** and VSS pins as possible.

Preliminary

7.2.6 test_clk_out

To be used only for testing the chip, **test_clk_out** is an output reference clock whose rising edge into the 40pF load nominally coincides with the start of an internal microcycle. The relationship between **pll_clk_in** and **test_clk_out** is deterministic following the second negation of **reset_in_l** after either powerup or change of the clock frequency ratio.

This pin should not be used to drive module level logic.

7.2.7 reset_in_l

reset_in_l is the master reset input for the chip and should be asserted when power is first applied to the chip. Low assertion of **reset_in_l** immediately initializes certain internal logic within the chip. Note that some internal state is not reset and must be handled by software when the chip boots. High negation of **reset_in_l** is synchronized to the internal clock and internal chip activity will start 31 cycles after negation.

7.2.8 irq<2:0>

These pins are not normally part of the clock system, but during reset these pins are tristated to receive the powerup values (through external resistors tied to VSS or VDD) to set the frequency ratio of the internal clock to the external reference clock. See Section 7.3 for further details.

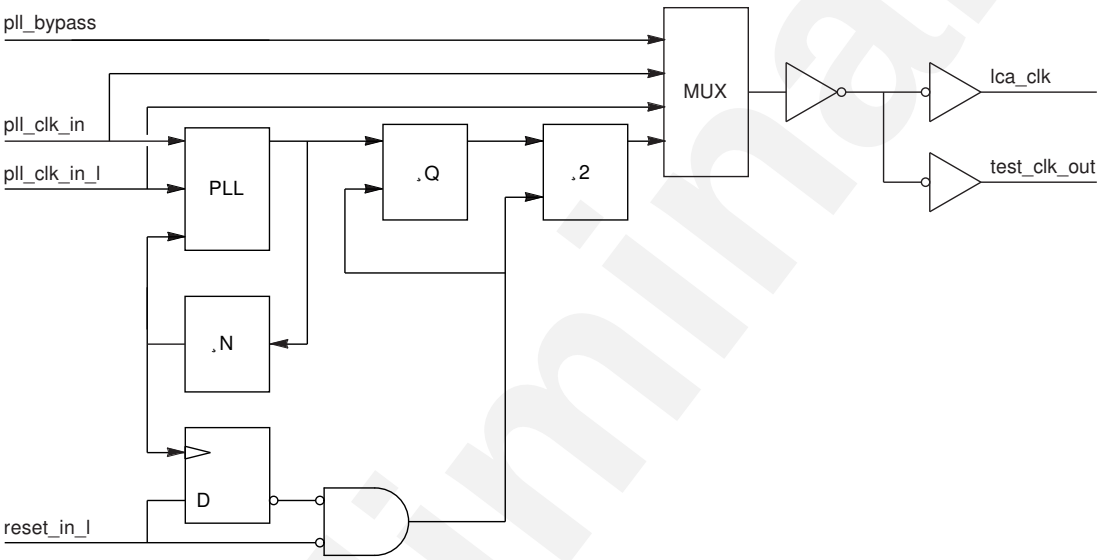
7.3 PLL System Overview

Figure 7–2 shows a simplified diagram of the internal clock generator. The MUX and PLL block comprises the voltage controlled oscillator (VCO) and related components necessary to implement the PLL. The divide-by-N block completes the PLL by providing a powerup-programmable feedback path from the VCO output to the phase detector input of the PLL. The ratio between the frequencies of the VCO output and the external clock reference driving **pll_clk_in** and **pll_clk_in_l** is N. This enables selection of different initial synthesis ratios for the PLL. The PLL output is further divided by Q, to complete the internal frequency generation process. The internal clock frequency will be a multiple of N/2Q times the external clock frequency. The final divide-by-2 provides a clean 50% duty cycle square wave to be driven on **lca_clk**, the internal clock node. Note that **test_clk_out** is simply driven off-chip from some point selected in the global buffering between the final MUX output and **lca_clk**. No attempt is made to control the skew between **lca_clk** and **test_clk_out** other than to nominally match rising edges in simulation.

Preliminary

The values supplied on the external pins **irq<2:0>** during reset assertion and the resulting N and Q values are shown in Table 7–1. These values are latched upon negation of **reset_in_l**. During and after chip reset, these values select the N and Q values indicated. The N divider starts without reset, so its starting state is UNPREDICTABLE.

Figure 7–2 PLL Clock Generator



Note

Maximum internal clock speed is determined by system (package and power) requirements. Operating frequencies (**lca_clk** speeds) from 2 to 9 times the selected oscillator or input clock frequency are supported. **pll_clk_in** could be **pci_clk_in** to obviate the need for separate oscillators.

For **pll_bypass** operation, the clock received at the input pin(s) is buffered to become the internal clock.

Preliminary

Table 7–1 Internal Clock programming

IRQ <2:0>	N	Q	Internal Clock Frequency ÷ External Clock Input Frequency	
			pll_bypass = 0	pll_bypass = 1
000	12	3	2	1
001	12	2	3	1
010	16	2	4	1
011	10	1	5	1
100	12	1	6	1
101	14	1	7	1
110	16	1	8	1
111	18	1	9	1

Preliminary

Preliminary

8
JTAG

8.1 JTAG Test Port

The chip contains boundary scan register and a serial scan test port that conforms to the IEEE STD. 1149.1 specification. Refer to that specification for a detailed description of the JTAG function. It consists of the five-wire Test Access Port (TAP), a TAP controller, an instruction register (IR) a bypass register, and a boundary scan register.

8.2 JTAG Pins

The JTAG pins are summarized in Appendix A. Functional descriptions of each pin are given below.

8.3 TAP Controller

The TAP Controller is a state machine which interprets IEEE 1149.1 protocols received on **tms** line and generates appropriate clocks and control signals for the testability features under its jurisdiction.

The state transitions in the controller are caused by the **tms** signal on the rising edge of **tck**. In each state, the controller generates appropriate clocks and control signals that control the operation of the testability features. Appropriate actions of the testability features are initiated on the rising edge of **tck** following the entry into a state.

8.4 Instruction Register

The Instruction Register (IR) resides on a scan path. Its contents are interpreted as test instructions and are used to select the testability modes and features. The Instruction Register consists of five bits, interpreted as shown in Table 8–1 to select and control the operation of the boundary scan register. During Capture-IR state, the shift register stage of IR is loaded with data 00001.

Preliminary

8.5 Bypass Register

The Bypass Register is a one bit shift register which provides a single-bit serial connection through the port (chip) when no other test path is selected.

The bypass register provides a one bit scan route though the chip during a scan-shift operation. It provides a means for effectively bypassing the chip's boundary scan during testing at module and system levels.

Table 8–1 JTAG Instruction Register

IR Contents	Instruction Name	Test Register Selected	Operation
00000	EXTEST	BSR	External test (drives pins from BSR)
00001	SAMPLE	BSR	Samples I/O
00010	Private/BSROSC	BSR	Ring Oscillates BSR
00011	Private/BSRDLY	BSR	Configures BSR for propagation delay measurement
00100	CLAMP	BPR	Drives pins from BSR Selects BPR for shifts
00101	HIGHZ	BPR	Tristates all output and I/O pins
00110	Private/IDR	IDR	Scans out ID register
00111	Private/SPARE		Reserved (selects IDR)
01000 through 11110	Private/ICMODE	BPR	Used for Icache initialization and internal chip testing. See Table 8–2.
11111	BYPASS	BPR	Selects BPR for shifts

In high-Z state all chip outputs and I/O pins, with the exception of **tdo**, are put into a high impedance (tristate) state.

Note

When the JTAG is in EXTEST or high-Z mode or CLAMP mode, the chip's internal reset is asserted and it remains in the reset state until those instructions are removed.

Preliminary

Table 8–2 JTAG ICMODE Functions

IR <4:3>	IR <2:0>	ICMODE Function
1x	000..011	Reserved
1x	100	Icache test read mode
1x	101	Icache test write mode
1x	110	Disable serial interface
1x	111	Enable SROM Icache initialization mode
10	xxx	Select synchronous mode of operation for IOC. IR<2:0> determine the Icache or test mode as above.
11	xxx	Select asynchronous (normal) mode of operation for IOC. IR<2:0> determine the Icache or test mode as above.

8.6 Boundary Scan Register

The boundary scan register primarily facilitates interconnection test on modules during module manufacturing and field service.

The boundary scan register is a single shift register formed by boundary scan cells placed at most of the chip's signal pins. The register is accessed via the JTAG port's **tdi** and **tdo** pins.

8.6.1 Boundary Scan Register Cells

On input-only pins the boundary scan cell basically consists of 1-bit shift register. The cell supports Sample and Shift functions.

On output-only pins, besides the shift register, the cell has an output multiplexer. The cell supports the following functions: Sample, Shift, Drive outputs.

On bidirectional pins, the cell is identical to the output-only pin cell except that it captures test data from the incoming data line. The cell supports Sample, Shift, Drive output and Hold output functions. It is used at all I/O pins.

Preliminary

8.6.2 Boundary Scan Register Organization

The boundary scan register on the chip is 180 bits long. Table 8–3 lists all the signal pins in the order of the scan chain, from **mem_dtoe_l** through **trst_l**. The first entry in the scan chain is **mem_dtoe_l** and when scanned out it is the last entry. Table 8–3 also shows the JTAG groups. A group is a collection of I/O pins (only) that can be simultaneously tristated by setting a bit in the appropriate group control register. The group control register is physically located in the chain as shown.

Table 8–3 Boundary Scan Order

1	**mem_dtoe_l**	90	Group Control Register 3
2	**vrefresh_l**	91	**ad0**
3	**vframe_l**	⋮	
4	**mem_dsf**	98	**ad7**
5	**mem_rasa_l3**	99	**c_be_l0**
⋮		⋮	
8	**mem_rasa_l0**	102	**c_be_l3**
9	**mem_rasb_l3**	103	**lock_l**
⋮		104	**irdy_l**
12	**mem_rasb_l0**	105	**trdy_l**
13	Group Control Register 1	106	**stop_l**
14	**mem_ecc0**	107	**devsel_l**
⋮		108	**perr_l**
21	**mem_ecc7**	109	**frame_l**
22	**mem_data31**	110	**par**
⋮		111	**gnt_l**
53	**mem_data0**	112	**req_l**
54	**mem_rd_oe**	113	**rst_1**
55	**mem_write_l**	114	**ad8**
56	**mem_wr_oe_l**	⋮	

(continued on next page)

Preliminary

Table 8–3 (Cont.) Boundary Scan Order

57	**mem_cas_l**	137	**ad31**
58	**mem_addr11**	138	Group Control Register 4
⋮		139	**mem_data32**
69	**mem_addr0**	⋮	
70	Group Control Register 2	170	**mem_data63**
71	**bc_index**	171	**reset_in_l**
72	**bc_idx_tag4**	172	**sromd**
⋮		173	**sromoe_l**
76	**bc_idx_tag0**	174	**sromclk**
77	**bc_parity**	175	**irq2**
78	**bc_dirty**	176	**irq1**
79	**bc_tag0**	177	**irq0**
⋮		178	**memreq_l**
86	**bc_tag7**	179	**memack_l**
87	**bc_oe_l**	180	**trst_l**
88	**bc_cs_l**		
89	**bc_we_l**		

8.7 Initialization

The TAP Controller and the Instruction Register's output latches are initialized when the TRST_L input is asserted. The TAP controller is forced to enter the Test-Logic-Reset state and the IR is forced to 11111 (BYPASS instruction, and the selected ICMODE is Serial ROM Init enabled, and IOC is in asynchronous (normal) mode of operation).

During Test-Logic-Reset state, all JTAG logic, including the boundary scan register, is in an inactive state. That is, the chip performs normal system functions. The boundary scan logic is set to a passive sample (observe) mode. The TAP controller leaves this state only when a JTAG test operation is desired and appropriate sequence is sent on **tms** and **tck** pins.

Preliminary

Preliminary

9

Icache Initialization and SROM Interface

This chapter provides details about the pins used and the operation of the serial ROM (SROM) interface.

9.1 Introduction

Rather than jumping to a fixed IO address upon reset, the microprocessor loads its initial I-stream from a compact SROM to start the bootstrap procedure. The SROM interface can also be used for chip and module level testing. In addition, using the PALcode, and the serial line transmit and receive registers (SL_XMIT, SL_RCV) a diagnostic terminal can be implemented.

9.2 Instruction Cache Initialization/SROM Interface

The microprocessor implements three Icache initialization modes to support chip and module level testing. The value placed in the JTAG IR (Instruction register) determines which of these modes is used after the microprocessor is reset, as shown in Table 9–1. IR bit 3 can be used to select IOC synchronous mode of operation independent of the Icache initialization and test modes controlled by the IR<2:0>.

When using the Icache test mode, after writing the Icache, the chip must be reset before using the Icache test read mode to properly reset internal pointers to the Icache.

Table 9–1 Icache Test Modes

IR <4:3>	IR <2:0>	Mode
1x	000..011	Reserved
1x	100	Icache test read

(continued on next page)

Preliminary

Table 9–1 (Cont.) Icache Test Modes

IR <4:3>	IR <2:0>	Mode
1x	101	Icache test write
1x	110	Disable serial interface
1x	111	Enable SROM Icache initialization mode

If the SROM Mode is selected, the microprocessor loads the contents of its internal Icache from an external SROM before executing its first instruction. The SROM may contain enough code to complete the configuration of the external interface, (for example, setting the timing on the external cache RAMs) and diagnose the path between the CPU chip and the real ROM. The microprocessor is in PALmode following the deassertion of **reset_l**–this gives the code loaded into the Icache access to all of the visible state within the chip.

Three signals are used to interface to the SROM.

- The **sromoe_l** output signal supplies the output enable to the SROM, serving both as an output enable and as a reset (refer to the SROM specifications for details).
- The **sromclk** output signal supplies the clock to the SROM that causes it to advance to the next bit.
- The **sromd** input signal allows the microprocessor to read the SROM data. In this mode the instruction cache is written at a rate of one bit each 126 CPU cycles.

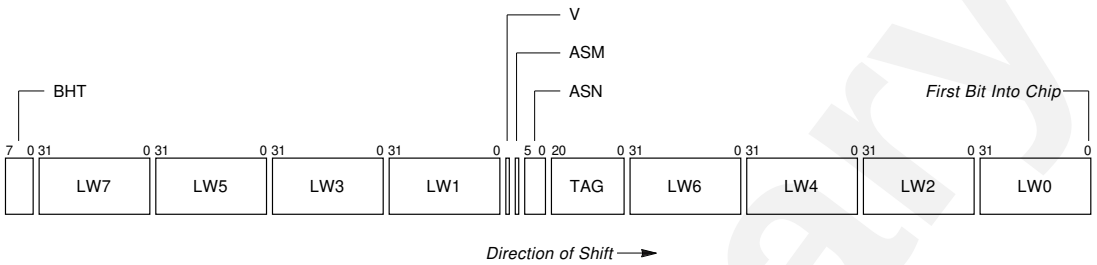
All Icache bits are loaded from the diagnostic interface, including each block's tag, ASN, ASM, valid and branch history bits. The Icache blocks are loaded in sequential order starting with block zero and ending with block 255. The order in which bits within each block are serially loaded is shown in Figure 9–1.

Note

In Figure 9–1, the most significant bit of each field is on the left. The serial chain shifts right, making bit 0 of LW0 the first bit shifted into the chip.

Preliminary

Figure 9–1 Icache Load Order



The valid and ASM bits in each cache block must be set. The tag field must also be written with zero. The value written into the branch history table (BHT) and address space number (ASN) fields are "don't cares".

When the microprocessor detects **reset_l** going high, based on the value in the JTAG instruction register, it loads bits from external SROM into its internal cache. Figure 9–2 shows the SROM timing.

First of Three Bit Samples

See Figure 9–2

1. When **reset_l** asserted, **sromoe_l** is deasserted and **sromclk** is assrted.
2. The microprocessor's internal reset signal remains asserted at least 20 CPU cyles after **reset_l** deasserts, when **sromoe_l** asserts.
3. The first rising edge of **sromclk** occurs 128 cycles after **sromoe_l** asserts, and every 126 cycles thereafter.
4. The microprocessor samples SROMD in the last half of each CPU cycle before the rising edge of **sromclk**.

This sequence continues until the Icache is loaded. There are 256 blocks in the Icache each containing 293 bits, or 75,008 bits in all, resulting in 75,008 rising edges of **sromclk**.

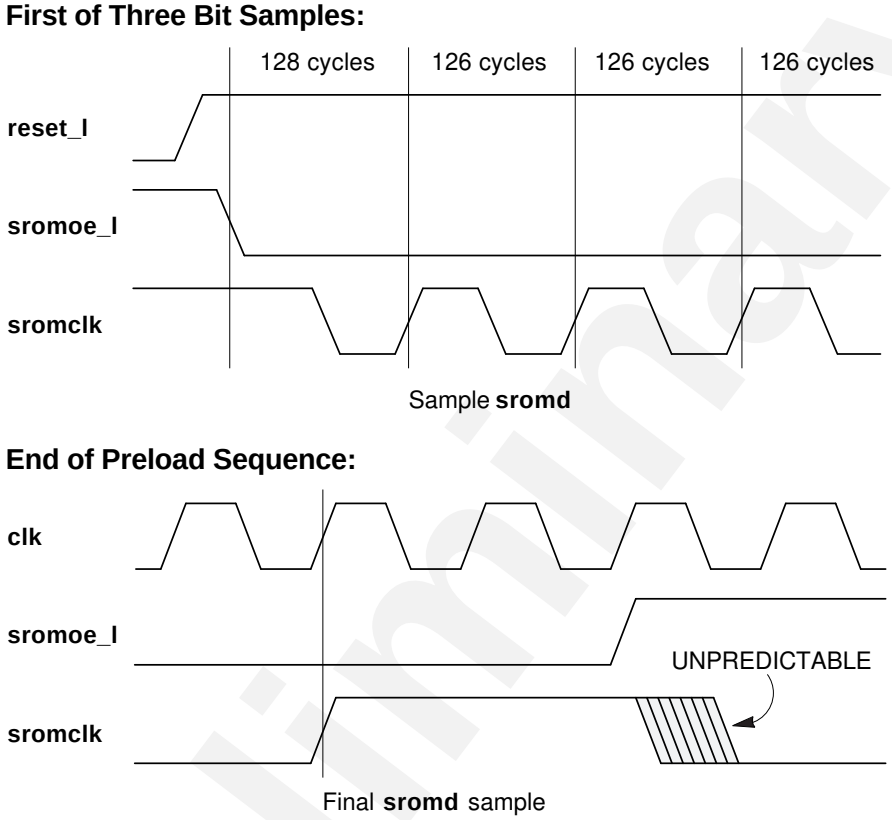
End of Preload Sequence

See Figure 9–2. At the end of the Icache preload sequence, the shaded area where **sromclk** goes low indicates UNPREDICTABLE behavior. **clk** refers to the microprocessor's internal CPU clock and is shown as cycle reference.

1. The microprocessor samples the final SROM bit when **sromclk** rises, as shown.

Preliminary

Figure 9–2 SROM Timing



2. Two CPU cycles later, the microprocessor deasserts **sromoe_l** and drives **sromclk** with the value from the TMT bit of the SL_XMIT IPR. Since this bit is not initialized by chip reset, the value driven onto **sromclk** is UNPREDICTABLE.

In addition to the SROM mode, the chip includes two test modes which allow chip tester hardware full read and write access to the Icache. Icache Test Write mode works exactly like SROM mode except that bits are loaded into the Icache at a higher rate. Icache Test Read mode allows the contents of the Icache to be read in a bit-serial manner from the **sromoe_l** pin. These two test modes are available only to chip test hardware.

Preliminary

9.3 Serial Line Interface

Once the data in the SROM has been loaded into the Icache, the three special signals become simple parallel I/O pins that can be used to drive a diagnostic terminal. When the SROM is not being read, the **sromoe_l** output signal is false. This means that the **sromoe_l** pin can be wired to the active high enable of an RS422 receiver driving onto SROMD and to the active high enable of an RS422 driver driving from **sromclk**. The CPU allows **sromd** to be read and **sromclk** to be written by PALcode; this is sufficient hardware support to implement a software driven serial interface.

Preliminary

Preliminary

A
Pin Summary

A.1 Memory Controller Pins

Table A–1 Memory Controller Pins

Name	Number	Type	Description	Value at Reset
mem_data<63:0>	64	I/O	Memory/Bcache data	Tristate
mem_ecc<7:0>	8	I/O	Memory/Bcache error correcting code	Tristate
mem_addr<11:0>	12	O	Row/column address, Bcache index	Driven, undefined
mem_write_l	1	O	Write enable	Driven, deasserted
mem_rasa_l<3:0> **mem_rasb_l<3:0>**	8	O	Row address strobes	Driven, deasserted
mem_cas_l	1	O	Column address strobe	Driven, deasserted
mem_dtoe_l	1	O	Data transfer/output enable	Driven, deasserted
mem_dsf	1	O	Disable special function	Driven, deasserted
mem_rd_oe	1	O	Memory read transceiver output enable	Driven, deasserted
mem_wr_oe_l	1	O	Memory write transceiver output enable	Driven, asserted
bc_tag<7:0>	8	I/O	Bcache tag	Tristate
bc_parity	1	I/O	Bcache tag parity	Driven, undefined
bc_index	1	O	Bcache index (bit 12)	Driven, undefined

(continued on next page)

Preliminary

Table A–1 (Cont.) Memory Controller Pins

Name	Number	Type	Description	Value at Reset
bc_idx_ tag<4:0>	5	I/O	Bcache index or tag	Tristate
bc_cs_l	1	O	Bcache chip select	Driven, asserted
bc_we_l	1	O	Bcache write enable	Driven, deasserted
bc_oe_l	1	O	Bcache output enable	Driven, asserted
bc_dirty	1	I/O	Bcache valid	Driven, undefined
vframe_l	1	I	Load video display pointer and load VRAM shift register	N/A
vrefresh_l	1	I	Increment video display pointer and load VRAM shift register	N/A

A.2 PCI Bus Pins

Table A–2 PCI Bus Signal Pins

Name	Number	Type	Description
ad<31:0>	32	I/O	PCI multiplexed address and data bus
c_be_l<3:0>	4	I/O	PCI multiplexed cycle command and byte enables
frame_l	1	I/O	PCI cycle frame
trdy_l	1	I/O	PCI target ready
irdy_l	1	I/O	PCI initiator ready
stop_l	1	I/O	PCI target stop
par	1	I/O	PCI parity data
perr_l	1	I/O	PCI parity error
devsel_l	1	I/O	PCI device select
req_l	1	O	PCI bus request, tristated during reset
gnt_l	1	I	PCI bus grant
rst	1	O	PCI reset

(continued on next page)

Preliminary

Table A–2 (Cont.) PCI Bus Signal Pins

Name	Number	Type	Description
lock_l	1	I	PCI lock. This pin is also used in testing the retry time counter during the test mode. This test mode can be set by programming the JTAG IR register (IR3 = 0). During the test mode, this pin selects which half of the 24-bit retry timeout counter. A high on this pin enables testing the lower half and a low on this pin tests the upper half.
pci_clk_in	1	I	PCI clock input
memreq_l	1	I	PCI request for memory access
memack_l	1	O	Memory access grant from IOC

A.3 Clock Pins

Table A–3 Clock Pins

Name	Number	Type	Description	Value at Reset
pll_clk_in	1	I	PLL clock input	N/A
pll_clk_in_l	1	I	PLL clock input low	N/A
pll_bypass	1	I	Select PLL bypass	N/A
pll_i_ref	1	I	Phase lock loop reference current	N/A
pll_filter	1	I	Phase lock loop low-pass filter capacitor	N/A
pll_5v	1	I	Phase lock loop voltage supply	N/A
test_clk_out	1	O	Output clock	Driven, clocking
reset_in_l	1	I	Master reset input	N/A

A.4 JTAG Pins

Table A–4 JTAG Pins

Name	Number	Type	Description	Value at Reset
tdi	1	I	Serial boundary scan data in	N/A
tdo	1	O	Serial boundary scan data out	Driven, undefined

(continued on next page)

Preliminary

Table A–4 (Cont.) JTAG Pins

Name	Number	Type	Description	Value at Reset
tms	1	I	JTAG test mode select	N/A
tck	1	I	Boundary scan clock	N/A
trst_l	1	I	JTAG TAP reset	N/A

A.5 Interrupt, SROM Interface, and INSTR_REF Pins

Table A–5 Interrupt, SROM Interface, and INSTR_REF Pins

Name	Number	Type	Description	Value at Reset
irq<2:0>	3	I	External interrupt request inputs. These pins are also used to program chip internal clock frequency during reset.	N/A
sromoe_l	1	O	Serial ROM output enable	Driven high
sromd	1	I	SROM data/receive serial data	N/A
sromclk	1	O	SROM clock/transmit serial data	Driven high
instr_ref	1	O	This pin has dual functions. During normal operation, it indicates the type of current access (instruction or data) on the memory data bus. A high on this pin indicates I-stream reference; a low on this pin indicates D-stream reference. During reset, the internal PCI clock is output on this pin for test purposes.	Driven, PCI clock

Preliminary

B
Internal Register Summary

B.1 Internal Processor Registers

Table B–1 Implementation-Specific Internal Processor Registers

Mnemonic	Register Name	Field*	Index*	Reset State
Instruction Fetch and Decode Unit Registers				
ASTER	Asynchronous system trap interrupt enable	IBX	18	Undefined
ASTRR	Asynchronous system trap request	IBX	14	Undefined
EXC_ADDR	Exception Address	IBX	4	Undefined
EXC_SUM	Exception Summary	IBX	10	Undefined
HIER	Hardware interrupt enable	IBX	16	Undefined
HIRR	Hardware interrupt request	IBX	12	Not applicable
ICCSR	Instruction cache control and status	IBX	2	Cleared except ASN, PC0, and PC1 bits.
ITBASM	Instruction translation buffer address space match	IBX	7	Not applicable
ITBIS	Instruction translation buffer initial state	IBX	8	Not applicable
ITB_PTE	Instruction translation buffer page table entry	IBX	1	Undefined
ITB_PTE_TEMP	Instruction translation buffer page table entry temporary	IBX	3	Undefined
ITBZAP	Instruction translation buffer ZAP	IBX	6	Not applicaple
PAL_BASE	PAL base address	IBX	11	Cleared
PS	Processor Status	IBX	9	Undefined
SIER	Software interrupt enable	IBX	17	Undefined

*HW_MFPR and HW_MTPR instruction fields: PAL, ABX, IBX, and Index (<7,6,5,4:0>).

(continued on next page)

Preliminary

Table B–1 (Cont.) Implementation-Specific Internal Processor Registers

Mnemonic	Register Name	Field*	Index*	Reset State
Instruction Fetch and Decode Unit Registers				
SIRR	Software interrupt Request	IBX	13	Undefined
SL_CLR	Clear serial line interrupt	IBX	19	Undefined
SL_RCV	Serial line receive	IBX	5	Undefined
SL_XMIT	Serial line transmit	IBX	22	Undefined
TB_TAG	Translation buffer tag	IBX	0	Undefined
Load and Store Unit Registers				
ABOX_CTL	Abox control	ABX	14	Cleared
ALT_MODE	Alternate processor mode	ABX	15	Undefined
CC	Cycle counter	ABX	16	Undefined
CC_CTL	Cycle counter control	ABX	17	Undefined
DC_STAT	Data cache status	ABX	12	Undefined
DTBASM	Data translation buffer address space match	ABX	7	Not applicable
DTB_CTL	Data translation buffer control	ABX	0	Undefined
DTBIS	Data translation buffer invalidate single	ABX	8	Not applicable
DTB_PTE	Data translation buffer page table entry	ABX	2	Undefined
DTB_PTE_TEMP	Data translation buffer page table entry Temporary	ABX	3	Undefined
DTBZAP	Data translation buffer ZAP	ABX	6	Not applicable
FLUSH_IC	Flush instruction cache	ABX	21	
FLUSH_IC_ASM	Flush instruction cache address space match	ABX	23	
MM_CSR	Memory management control and status	ABX	4	Undefined
VA	Virtual address	ABX	5	Undefined
PAL Temporary Registers				
PAL_TEMP<31:0>	PAL_TEMP internal processor	PAL	31..0	Undefined

*HW_MFPR and HW_MTPR instruction fields: PAL, ABX, IBX, and Index (<7,6,5,4:0>).

Preliminary

B.2 Memory Controller Registers

Table B–2 Memory Controller Registers

Mnemonic	Register Name	Address (Hexadecimal)	Reset State
BCR0	Bank configuration 0	1 2000 0000	BAV bit = 0, all other bits are UNDEFINED
BCR1	Bank configuration 1	1 2000 0008	BAV bit = 0, all other bits are UNDEFINED
BCR2	Bank configuration 2	1 2000 0010	BAV bit = 0, all other bits are UNDEFINED
BCR3	Bank configuration 3	1 2000 0018	BAV bit = 0, all other bits are UNDEFINED
BMR0	Bank mask 0	1 2000 0020	UNDEFINED
BMR1	Bank mask 1	1 2000 0028	UNDEFINED
BMR2	Bank mask 2	1 2000 0030	UNDEFINED
BMR3	Bank mask 3	1 2000 0038	UNDEFINED
BTR0	Bank timing 0	1 2000 0040	UNDEFINED
BTR1	Bank timing 1	1 2000 0048	UNDEFINED
BTR2	Bank timing 2	1 2000 0050	UNDEFINED
BTR3	Bank timing 3	1 2000 0058	UNDEFINED
GTR	Global timing	1 2000 0060	REN bit = 0, all other bits are UNDEFINED
ESR	Error status	1 2000 0068	UNDEFINED
EAR	Error address	1 2000 0070	UNDEFINED
CAR	Cache control	1 2000 0078	BCE bit = 1, Bcache size = 0 (64K bytes) and Bcache read speed field = 0 (that is, speed is three cycles). All other bits are UNDEFINED.
VGR	Video and graphics control	1 2000 0080	UNDEFINED
PLM	Plane mask	1 2000 0088	UNDEFINED
FOR	Foreground	1 2000 0090	UNDEFINED

B.3 I/O Controller Registers

Preliminary

Table B–3 I/O Controller Registers

Mnemonic	Register Name	Address	Reset State and Required Initialization
IOC_HAE	Host address extension	1 8000 0000	UNDEFINED
IOC_CONF	Configuration cycle type	1 8000 0020	UNDEFINED
IOC_STAT0	Status 0	1 8000 0040	UNDEFINED
IOC_STAT1	Status 1	1 8000 0060	UNDEFINED
IOC_TBIA	Translation buffer invalidate all	1 8000 0080	The TLB is undefined and must be initialized by software.
IOC_TB_ENA	Tranlsation buffer enable	1 8000 00A0	UNDEFINED
IOC_SFT_RST	PCI soft reset	1 8000 00C0	The soft reset bit, and consequently the PCI **rst_l** signal, will be asserted during and after reset until modified by writing to this register.
IOC_PAR_DIS	Parity disable	1 8000 00E0	The parity disable bit is cleared to enable parity checking and reporting after reset.
IOC_W_BASE0	Window base 0	1 8000 0100	UNDEFINED. SROM bootcode must initialize the WEN and SG bits.
IOC_W_BASE1	Window base 1	1 8000 0120	UNDEFINED.
IOC_W_MASK0	Window mask 0	1 8000 0140	UNDEFINED
IOC_W_MASK1	Window mask 1	1 8000 0160	UNDEFINED
IOC_T_BASE0	Translated base 0	1 8000 0180	UNDEFINED
IOC_T_BASE1	Translated base 1	1 8000 01A0	UNDEFINED
IOC_TB_TAG0	Translation buffer tag 0	1 8100 0000	UNDEFINED
IOC_TB_TAG1	Translation buffer tag 1	1 8100 0020	UNDEFINED
IOC_TB_TAG2	Translation buffer tag 2	1 8100 0040	UNDEFINED
IOC_TB_TAG3	Translation buffer tag 3	1 8100 0060	UNDEFINED
IOC_TB_TAG4	Translation buffer tag 4	1 8100 0080	UNDEFINED
IOC_TB_TAG5	Translation buffer tag 5	1 8100 00A0	UNDEFINED
IOC_TB_TAG6	Translation buffer tag 6	1 8100 00C0	UNDEFINED
IOC_TB_TAG7	Translation buffer tag 7	1 8100 00E0	UNDEFINED
IOC_IACK_SC	Interrupt vector and special cycle	*	UNDEFINED

*Any quadword aligned address in the range 1A0000000..1BFFFFFE0.

Programming Note

The IOC register address (34-bit) range is 1 8000 0000 through 1 BFFF FFFF and the IOC registers are sparsely populated in this range. Although the IOC internally decodes only a subset of the 34-bit address to access the IOC registers, it is recommended that programs only use

Preliminary

the addresses specified in Table B–3 to ensure correct operation in the future implementations.

Preliminary

Preliminary

Preliminary

Preliminary

Preliminary